Tema: Videojuegos, procesado de señal, rehabilitación de la voz, Android

Título: Desarrollo de videojuegos para rehabilitación de la voz en entornos móviles

Autor: John Jeremy Ireland

Titulación: Imagen y sonido

Tutor: Juan Ignacio Godino Llorente

Departamento: ICS

Director: César Sanz Álvaro

Tribunal

Presidente: Juan Manuel Meneses Chaus

Vocal: Juan Ignacio Godino Llorente

Vocal Secretario: Rubén Fraile Muñoz

Fecha de lectura: Septiembre 2014

Índice de contenidos

Resumen

Este proyecto consiste en crear una serie de tres pequeños videojuegos incluidos en una sola aplicación, para plataformas móviles Android, que permitan en cualquier lugar entrenar la estética de la voz del paciente con problemas de fonación. Dependiendo de los aspectos de la voz (sonidos sonoros y sordos, el pitch y la intensidad) a trabajar se le asignará un ejercicio u otro. En primer lugar se introduce el concepto de rehabilitación de la voz y en qué casos es necesario. Seguidamente se realiza un trabajo de búsqueda en el que se identifican las distintas plataformas de desarrollo de videojuegos que son compatibles con los sistemas Android, así como para la captura de audio y las librerías de procesado de señal. A continuación se eligen las herramientas que presentan las mejores capacidades y con las que se va a trabajar. Estas son el motor de juego Andengine, para la parte gráfica, el entorno Java específico de Android, para la captura de muestras de audio y la librería JTransforms que realiza transformadas de Fourier permitiendo procesar el audio para la detección de pitch. Al desarrollar y ensamblar los distintos bloques se prioriza el funcionamiento en tiempo real de la aplicación. Las líneas de mejora y conclusiones se comentan en el último capítulo del trabajo así como el manual de usuario para mayor comprensión.

Abstract

The aim of this project is to create an application for mobile devices which includes three small speech therapy videogames for the Android OS. These videogames allow patients to train certain voice parameters (such as voice and unvoiced sounds, pitch and intensity) wherever they are and whenever they have the need. First, an overview of the concept of voice rehabilitation and its usefulness for patients with speech disorders is given. Next, a study is presented in order to identify the most suitable video game engine for the Android OS, the best possible way to capture audio from the device and also the audio processing library which will best combine with them. Then, the selected tools are identified. Andengine has been chosen as the game engine, Android's Java framework is preferred for audio capture and the fast Fourier transform library, JTransforms, is designated for pitch detection. When developing and assembling the three tools, the overriding concern is performance in real time. Finally, conclusions are presented, together with suggestions for possible improvements and a user's manual.

Agradecimientos

En primer lugar deseo agradecer a mis padres por todo el apoyo y confianza que me han dado a lo largo de mis estudios y las facilidades que me han dado siempre para acometer mis proyectos.

También quiero agradecer a mi hermana por sus ánimos y su interés en mis estudios por mucho que vivamos alejados.

Doy las gracias a mis amigos de la universidad y a los de fuera de ella por todos los buenos momentos que hemos pasado y que han hecho muy amena esta etapa universitaria.

Finalmente deseo agradecer a mi tutor Juan Ignacio Godino y a Rubén Fraile Muñoz por sus correcciones y sugerencias.

Glosario

A

ADT: Android Development Tools, *Herramientas de desarrollo Android*
API: Application Programming Interface, *Interfaz de programación de aplicaciones*

B

Buffer: Búfer de datos, espacio de memoria en el que se almacenan datos

C

Callback: En programación, devolución de llamada o retrollamada

D

DCT: Discrete Cosine Transform, *Transformada discreta del coseno*
DDMS: Dalvik Debug Monitor Service, *Servicio de monitorización de depuración Dalvik*
DFT: Discrete Fourier Transform, *Transformada discreta de Fourier*
DHT: Discrete Hartley Transform, *Transformada discreta de Hartley*
Dithering: Es una técnica que se usa en procesado de imágenes para crear una ilusión de profundidad de color cuando se tiene una paleta reducida de colores [32]
Disfonía: Trastorno de la fonación [30]
DST: Discrete Sine Transform, *Transformada discreta del seno*

E

Edema: Acumulación excesiva de líquido en algún órgano o tejido del cuerpo [29]

F

Feedback: Realimentación
FFT: Fast Fourier Transform, *Transformada rápida de Fourier*
Fonación: Emisión de la voz o de la palabra
Frame: Imagen particular dentro de una sucesión de imágenes que componen una animación

Framework: Entorno

G

Getter: Método de acceso que permite en la aplicación desarrollada obtener la escena actual
Glotis: Espacio alargado de la laringe o abertura limitada lateralmente por los bordes de las cuerdas vocales inferiores [29]
GUI: Graphic User Interface, *Interfaz gráfica de usuario*

I

IDE: Integrated Development Environment, *Entorno de desarrollo integrado*
Impostación: Emisión de la voz en toda su plenitud, sin vacilación ni temblor [31]

J

Java: Un lenguaje de programación orientado a objetos y basado en clases

L

Legato: Ejecución de una serie de notas musicales consecutivas sin pausas ni interrupciones entre unas y otras [29]
Lua: Lenguaje de programación compacto usado en múltiples plataformas de videojuegos

N

Nódulo: Lesión elemental de la piel que se localiza en el tejido celular subcutáneo [30]

O

OpenGL: *Open Graphics Library*, es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones con gráficos 2D y 3D

P

Parallax scrolling: Desplazamiento diferencial, técnica que da sensación de profundidad

en videojuegos en 2D
Pitch: Tono o frecuencia fundamental de una señal de audio
Plugin: Programa anexo a otro para aumentar sus funcionalidades
Pólipo: Masa de células que se forma y crece en los tejidos que cubren el interior de algunos conductos del cuerpo que se comunican con el exterior [29]
Praxia: Sistema de movimientos coordinados en función de un resultado o de una intención [32]
Psicógeno: Que tiene origen psicológico y es patológico [29]

Q

Quiste: Cavidad patológica, delimitada por una membrana propia, y que contiene líquido en su interior [30]

S

SDK: Software Development Kit, *Kit de desarrollo de software*
Setter: Método de acceso que permite en la aplicación desarrollada establecer una escena
Smartphone: Teléfono inteligente
SPL: Sound Pressure Level, *Nivel de presión sonora*
Splash scene: Pantalla de arranque de una aplicación o programa informático
Sprite: Mapa de bits en 2D que se coloca en la pantalla según unas coordenadas
Staccato: Forma de ejecución en que se marca la separación entre notas [31]

U

USB: Universal Serial Bus, *Bus de serie universal*, es un protocolo que sirve para conectar distintos periféricos entre sí

W

Workspace: Área de trabajo en un software

X

XML: Extensible Markup Language, Lenguaje de marcas para almacenar datos de forma legible

Capítulo 1: Introducción

El trabajo realizado en éste proyecto forma parte de otro más amplio en el que la aplicación genera unos resultados para cada ejercicio. Estos resultados son supervisados por un profesional médico que a continuación trata al paciente de forma específica. Este proyecto está centrado exclusivamente en la captura y análisis del audio recogido por el dispositivo móvil del paciente así como en la interfaz gráfica que emite un *feedback* en función de las características del sonido emitido.

1.1. Marco tecnológico

Las nuevas aplicaciones disponibles para dispositivos móviles están permitiendo a los usuarios acceder y supervisar contenidos sobre su salud y bienestar. Gracias a estas herramientas de gran utilidad, los usuarios pueden acceder desde sus *smartphones* a información para primeros auxilios, controlar una enfermedad como la diabetes, la presión arterial, el peso corporal, las calorías que consumen, diagnósticos y tratamientos hasta cómo iniciar una rutina de ejercicios para mantener el cuerpo en forma [1].

Cómo no podía ser de otra manera, también se han desarrollado aplicaciones para el tratamiento e intervención de los trastornos del habla y voz. Se denominan visualizadores del habla ya que utilizan animaciones e imágenes como respuesta a la emisión de sonido y como elementos fundamentales para la retroalimentación y motivación del paciente [2].

1.2. Objetivos

La finalidad del proyecto es crear una aplicación para dispositivos Android que permita rehabilitar la estética de la voz de pacientes que sufren problemas de fonación. Para ello, se les entrena proporcionándoles un instrumento visual que puede ser usado en cualquier lugar y que les permite trabajar distintos aspectos de la voz a través de videojuegos. Esto hace que los ejercicios se conviertan en una actividad amena y práctica ya que la aplicación se usa fácilmente y de forma intuitiva. Los tres parámetros de la voz tratados y que se entrenan en cada uno de los juegos son:

- La fonación de sonidos sordos y sonoros
- La intensidad
- El tono ó *pitch*

1.3. Estructura de la memoria

La memoria del proyecto consta de seis capítulos que se enumeran y comentan brevemente a continuación:

La introducción establece el marco de trabajo y tecnológico del proyecto así como los objetivos que pretende cumplir. Explica con carácter general en qué consiste la aplicación desarrollada y cómo se ha estructurado esta memoria.

El segundo capítulo introduce el concepto de la rehabilitación de la voz para voces patológicas. Se describe la anatomía funcional de la voz, las afecciones vocales más comunes, así como las técnicas de rehabilitación empleadas según los casos.

La tercera parte, llamada *Estado de la cuestión*, da una visión general del marco tecnológico actual. Se diferencian dos tipos de aplicaciones para la rehabilitación de la voz en dispositivos móviles. Son presentadas distintas aplicaciones ya existentes de cada tipo. Además se exponen herramientas cuyo uso ha sido contemplado para el desarrollo del proyecto pero que finalmente han sido desechadas.

A continuación se encuentra la parte sustancial de la memoria, *Descripción de la solución propuesta*, que detalla cómo ha sido realizada la aplicación en cuestión desglosando el trabajo en distintos apartados. Primero explicando las herramientas seleccionadas para el trabajo y a continuación la estructura general del *software* desarrollado.

En *Conclusiones y líneas de trabajo futuras*, se da una visión general del trabajo realizado recordando la problemática y la solución propuesta. También se proponen posibles mejoras de la aplicación para futuras líneas de trabajo.

Finalmente, en los apéndices se comentan el manejo básico del *software* y su funcionamiento para uso posterior.

Capítulo 2: Producción, patologías y rehabilitación de la voz

En este capítulo se explican distintas técnicas y tratamientos para rehabilitar la voz de un paciente que sufre de una disfonía en particular. Para ello es necesario previamente familiarizarse con los patrones normales de la voz, la estructura del aparato fonador y su funcionamiento. Esto sirve para reconocer e identificar las características de la voz patológica, y para diagnosticarla y tratarla de manera adecuada.

2.1. Producción de la voz

2.1.1. Anatomía funcional de la voz

Para que se genere la voz en un individuo es necesario que se produzca una toma de aire o inspiración que llene de aire los pulmones y luego salga a través de la espiración. Durante la espiración, el aire pasa por unos órganos que harán posible la producción de la voz.

Los órganos que intervienen en la producción de la voz se pueden clasificar en tres grupos distintos:

- Aparato respiratorio
- Laringe
- Cavidades supraglóticas

A continuación se comentan los órganos que constituyen cada uno de estos conjuntos así como su funcionamiento dentro del proceso de producción de voz.

- Aparato respiratorio: Este conjunto está compuesto principalmente por la cavidad nasal, la faringe, la epiglotis, la laringe, la tráquea, los bronquios, los pulmones y el diafragma, tal y como se aprecia en la Figura 1.

  El diafragma, situado debajo de los pulmones, es el músculo principal de la inspiración. Durante la inspiración se contrae, descendiendo, y durante la espiración se relaja ascendiendo.

  La tráquea se sitúa anterior al esófago. Se extiende entre la laringe y los

bronquios principales, derecho e izquierdo, donde se bifurca. Su función es la de conducir el aire hacia los pulmones o fuera de ellos.

La epiglotis es una válvula que permanece abierta cuando se respira y se cierra cuando se traga para impedir que los alimentos o la saliva entren a la laringe y al resto del sistema respiratorio [3].

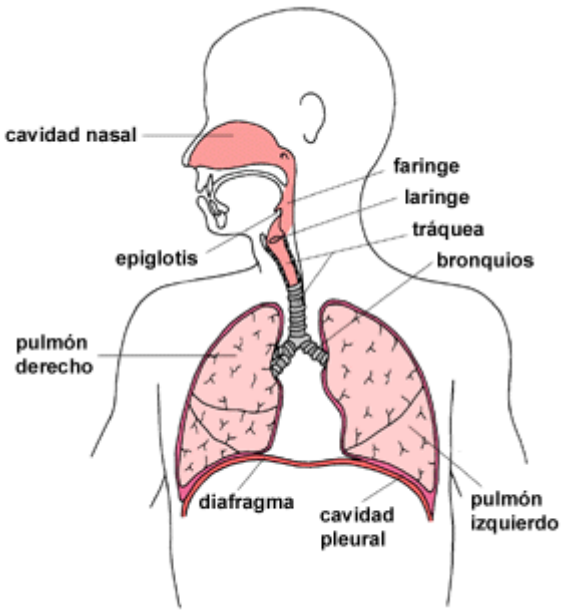

Figura 1: Aparato respiratorio humano, tomada de [4]

En la inspiración, la capacidad torácica aumenta, el diafragma desciende y al ensancharse los pulmones, se produce una reducción de la presión en relación con el aire exterior y el aire es inspirado hacia el interior a través de la nariz y la boca. El aire pasa por la tráquea hacia los pulmones.

Para la espiración se crea el proceso contrario al anteriormente comentado. El diafragma vuelve a su posición inicial. Se crea un centro de alta presión en los pulmones, lo que impulsa el aire a salir por la tráquea y de esta a la laringe, en la que se encuentran las cuerdas vocales. El flujo de aire continúa su camino por la faringe hacia la nariz y la boca, hasta llegar a los labios.

El propósito principal de la respiración es la de proveer al cuerpo de oxígeno, sin embargo, el aire que entra y sale también es combustible para la producción de la voz. A continuación se comenta el funcionamiento y composición de la laringe [4].

- Laringe: La laringe tiene la función de proteger las vías respiratorias y de producir los sonidos bajo la acción del aire espiratorio. Se sitúa en la parte medial y anterior del cuello, por delante de la faringe. Comunica a través de la faringe con la cavidad bucal

y nasal, y caudalmente con la tráquea, tal y como se aprecia en la Figura 2.

La laringe contiene los pliegues vocales denominados comúnmente cuerdas vocales. Durante la fonación, las cuerdas vocales actúan como un transductor que convierte la energía aerodinámica, generada por el aparato respiratorio, en energía acústica radiada a los labios, que se percibe como voz. [5 Capítulo "Fisiologia de la fonación" pág. 56][1]

La producción de voz o fonación se realiza en las cuerdas vocales a causa del denominado *efecto Bernoulli*. A medida que el aire intrapulmonar es expulsado se produce un aumento progresivo de la presión subglótica. Cuando la presión es superior a la de cierre de los pliegues vocales, éstos son obligados a separarse y el aire sale con fuerza. En este momento se produce un descenso de la presión y las cuerdas vocales vuelven a cerrarse. Este fenómeno se repite de forma rápida lo que conlleva a la vibración de las cuerdas vocales y por lo tanto la producción de voz [3].

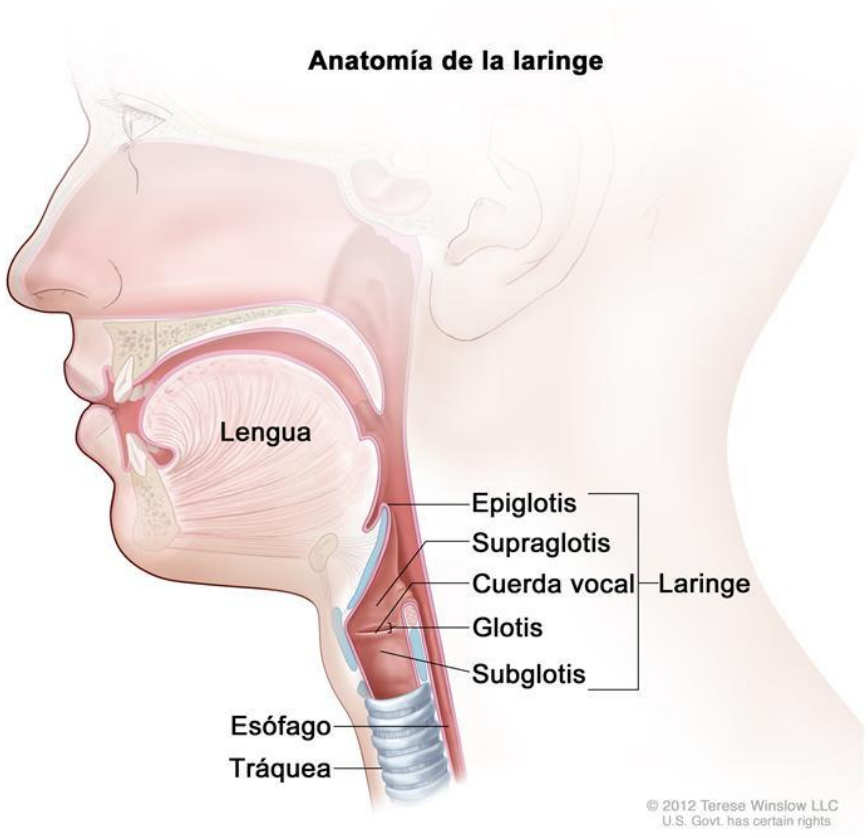

Figura 2: Anatomía de la laringe, tomada de [6]

- Cavidades supraglóticas: Las cavidades situadas por encima de los pliegues vocales (cuerdas vocales) actúan como cajas de resonancia de la voz. Se distinguen la boca, la faringe y las fosas nasales (Figura 3).

Estas estructuras anatómicas constituyen los resonadores que enriquecen, amplifican, sonorizan y matizan el sonido generado en la glotis. Algunos de estos resonadores son estructuras fijas (como las fosas nasales que no pueden modificar su forma o volumen) y otras pueden modificar su configuración (como la boca o la faringe) para conseguir las características acústicas del sonido que se pretenda emitir.

La boca se modificará en función de la apertura mandibular y de la posición de la lengua y labios. La faringe cambia su morfología en función del desplazamiento de la laringe, la lengua y el velo del paladar [3].

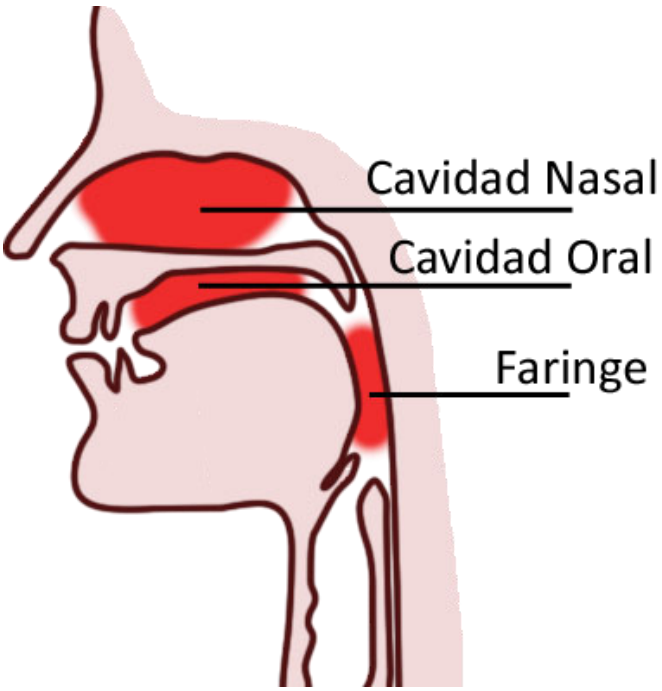

Figura 3: Aparato resonador, tomada de [7]

2.1.2. Características acústicas generales de la voz

El sonido vocal que sensorialmente percibimos es lo que conocemos como tono. Para caracterizarlo se definen unas propiedades fundamentales:

- Frecuencia: La variación de la frecuencia se consigue mediante el ajuste que realiza el sistema nervioso central y periférico, tanto en la región glótica como en la cavidad de resonancia. En la glotis, la frecuencia fundamental de oscilación de las cuerdas vocales puede controlarse mediante los cambios de longitud, masa y elasticidad de los planos que forman las cuerdas vocales ante la acción muscular. Cuando la frecuencia fundamental es alta, la mucosa se encuentra estirada y tensa, por lo que la ondulación es mínima y de muy limitado desplazamiento. El efecto contrario, es decir, un acortamiento de la cuerda vocal y una disminución de la tensión, se traduce en un

descenso de la frecuencia fundamental de vibración o *pitch*.

- Intensidad: El control del volumen, o intensidad, está relacionado con el flujo de aire y la presión con que éste se expulsa, es decir, con la potencia aerodinámica que se invierta en el proceso. Cuanto más intensa sea la fuerza, mayor es la presión subglótica y la resistencia de la válvula glótica al paso del aire, y con ello la intensidad.

- Extensión: Al conjunto de frecuencias que pueden ser emitidas por la laringe se le denomina extensión de la voz. En condiciones normales, la extensión para las voces masculinas oscila en un rango de frecuencias entre 80 y 700 Hz, y para las femeninas en un rango entre 140 y 1110 Hz.

- Timbre: El timbre es la propiedad de la voz que permite distinguir dos notas de igual frecuencia e intensidad emitidas por instrumentos musicales distintos, o diferenciar dos voces pertenecientes a personas distintas. El timbre depende de los formantes y de las dimensiones físicas del tracto vocal, de la frecuencia fundamental y de la intensidad. [5 Capítulo "Notas sobre acústica vocal" pág. 99-100]

2.2. Patologías de las cuerdas vocales

Se considera que hay un trastorno de la voz o disfonía cuando su timbre, tono, intensidad o flexibilidad difieren de los de las voces de las demás personas del mismo sexo, edad y grupo cultural. En la actualidad no existe una nomenclatura estándar para los trastornos de la voz ni para la patología de las cuerdas vocales. A pesar de ello se ha clasificado las distintas disfonías en los grupos siguientes [5 Capítulo "Voz normal y clasificación de las disfonías" pág. 238]:

- Patologías orgánicas: La disfonía orgánica consiste en trastornos de la voz que son debidos a alteraciones anatómicas en los órganos fono-articulatorios (esencialmente la laringe y cuerdas vocales) responsables de la producción de la voz.

- Lesiones mínimas asociadas: Son lesiones benignas de las cuerdas vocales que son frecuentemente diagnosticadas en la edad adulta. La mayoría de las lesiones en las cuerdas vocales son benignas, y en general deberán ser tratadas de manera conservadora, lo que significa agotar primero todos los recursos terapéuticos no quirúrgicos y sólo recurrir a la cirugía cuando los aspectos funcionales clave sigan alterados.

- Lesiones funcionales: El término disfonía funcional se utiliza generalmente para hacer referencia a un trastorno de la voz que no se debe a una enfermedad orgánica identificable. Describe un grupo amplio y diverso de perfiles clínicos que implican una alteración de la función vocal por un uso inadecuado con habituación de músculos voluntarios del complejo muscular oral y faringolaríngeo, del sistema de la

respiración y de grupos musculares posturales más generales [8 Capítulo "Clasificación de los trastornos de la voz por uso muscular inadecuado" pág. 56].

- Disfonías psicógenas: Es la alteración de la voz producida por un trastorno psicológico. Se trata de una alteración muy poco frecuente, en la que no existe lesión anatómica o neurológica, sino que es resultado de un proceso de inhibición psicológica, con un comienzo brusco.

2.3. Rehabilitación de la voz

2.3.1. Principios e indicaciones de la terapia vocal

El tratamiento de las alteraciones de la voz es múltiple, pero el de la disfonía se basa esencialmente en tres pilares fundamentales: fármacos, cirugía y rehabilitación. No hay método mejor que otro y no son opciones excluyentes, sino que los factores implicados en la génesis del problema determinarán la organización del proceso terapéutico.

El tratamiento de interés en este trabajo son las técnicas rehabilitadoras que están concebidas para eliminar la vocalización incorrecta y restablecer una voz eficiente. Ramig y Verdolini proponen cuatro indicaciones para el tratamiento vocal:

- Indicación absoluta con el objetivo de resolver el trastorno vocal cuando los tratamientos quirúrgicos o farmacológicos no están indicados

- Como tratamiento inicial en aquellos casos en que puede evitarse el tratamiento médico o quirúrgico incluso aunque estuviera indicado.

- Antes o después del tratamiento quirúrgico para maximizar la voz a largo plazo.

- Como tratamiento preventivo para preservar la salud vocal

A continuación se enumeran distintas situaciones en las que se utiliza la terapia vocal para mejorar la calidad de la voz del paciente que sufre una disfonía en particular.

- Disfonía por tensión muscular: La reeducación vocal en estos casos enseña a evitar esfuerzos musculares inapropiados, que impiden un buen rendimiento vocal, y lograr una mejor emisión de la voz.

- Disfonía funcional del adolescente: Durante la muda de la voz en el varón se busca la estabilidad en la emisión, y para ello se emplean técnicas respiratorias y de impostación.

- Lesiones benignas: La terapia vocal resulta eficaz en una gran variedad de lesiones

benignas en las cuerdas vocales (nódulos vocales, edemas, pólipos laríngeos, quistes). Se trata de un tratamiento efectivo y no traumático, sirve para eliminar de manera permanente los patrones nocivos causantes de la lesión y puede resultar menos costoso que la cirugía.

- Parálisis recurrencial unilateral: El síntoma predominante de esta patología, que afecta los nervios y cuerdas vocales, es la disfonía por lo que se recomienda un tratamiento rehabilitador. En algunos casos de lesión del nervio se logra una atenuación de los síntomas disfónicos y se recupera la voz normal hasta hacer innecesaria la cirugía [5 Capítulo "Indicaciones y límites de la terapia vocal" pág. 442-445].

2.3.2. Perfeccionamiento vocal

La educación vocal debe conseguir que el sujeto llegue a conocer su voz y logre automatizar los procesos normales de fonación, para mejorar la eficacia comunicativa y evitar la aparición de afecciones laríngeas. Ésta es la mejor forma de prevención y la pauta más importante de higiene vocal.

El entrenamiento vocal debe buscar el mínimo esfuerzo y el máximo rendimiento de la voz, lo cual se consigue mediante una función respiratoria y vocal correcta, una gama tonal óptima, el enriquecimiento de la voz con armónicos y la mejora de la articulación.

Este entrenamiento consta de diversos pasos, que son los pilares de la rehabilitación tradicional:

- Relajación: La voz se produce de manera más saludable cuanto menos esfuerzo se realiza para emitirla. Los objetivos del trabajo de relajación en el entrenamiento vocal son:

  - Eliminar el trabajo muscular excesivo
  - Disociación muscular
  - Lograr el tono muscular adecuado para cada actividad

  La fonación implica un gran trabajo muscular; hablar y relajar son incompatibles. Ello no significa que todas las estructuras deban trabajar con su máxima tonicidad, sino que los ajustes musculares deben controlarse según la situación de comunicación.

  Se realizan ejercicios de relajación de la cabeza, hombros, mandíbula, labios y lengua a los pacientes según sus necesidades.

- Respiración: Los ejercicios respiratorios son uno de los pilares sobre los cuales debe basarse toda rehabilitación de la voz hablada o cantada. El aire es la materia prima de

la voz; sin aire, no hay sonido. En la medida en que se domine ese aire, se dominará la voz.

El trabajo respiratorio en el entrenamiento vocal tiene como objetivo:

- Lograr una inspiración silenciosa y rápida
- Modificar el tipo respiratorio, buscando el costodiafragmático
- Controlar el volumen y el flujo espiratorio para lograr una emisión menos forzada
- Controlar la espiración en relación con la resistencia glótica
- Evitar la acción de los músculos accesorios de la respiración
- Lograr una correcta dosificación del aire espirado

Los ejercicios respiratorios consisten principalmente en enseñar al paciente el tipo de respiración costodiafragmática que consiste en intentar que el diafragma baje abombando el abdomen hacia adelante y abrir las costillas hacia los lados. Se pide al paciente que realice inspiraciones nasales normales y que comience sacando el aire con una /s/ sostenida sin gastarlo todo. Algunos ejemplos de ejercicios son espirar con /f/ sostenida o entrecortada, espirar con vocales sin voz o con /ch/ entrecortadas.

Estos ejercicios sirven para ayudar al paciente a ser consciente de la respiración para poder modificar hábitos incorrectos que luego automatizará con la práctica.

- Resonancia: La caja de resonancia de nuestra voz es el tracto vocal que va desde las cuerdas vocales hasta los labios. La boca, que está constituida por una bóveda inmóvil y por partes móviles que son los órganos activos de la articulación (labios, lengua y paladar blando), desempeña un importante papel en la resonancia. De cómo se coloquen estos órganos y del espacio que se cree en dicha cavidad depende la calidad y la cantidad de los armónicos que se generen y, por ende, la calidad vocal que se produzca.

  En el entrenamiento de la resonancia se intenta obtener el máximo aprovechamiento de los resonadores naturales. La ejercitación sonora debe realizarse partiendo del tono habitual del paciente. Se trabaja con un teclado partiendo de dicho tono realizando escalas ascendentes y descendentes, ampliando la extensión vocal de acuerdo con las posibilidades de cada paciente y su patología.

  Algunos de los ejercicios consisten en:

  - Consonantes nasales o laterales /m/, /n/, /l/
  - Paso de /n/ a /a/ con lengua baja

- /i/ en un tono medio
- Consonantes posteriores, como /k/ y /g/
- Bostezos

- Impostación: En esta etapa se enseña al paciente una técnica vocal correcta que le permita emitir una voz sin esfuerzo, con un adecuado rendimiento, y restablecer los parámetros acústicos de la frecuencia, timbre e intensidad perdidos por la disfonía.

  La impostación cantada o entonada sirve para tomar mayor conciencia de los armónicos, conseguir una mejor adaptación de los resonadores, colocar mejor la voz, dominar la salida del aire y favorecer la proyección vocal. Durante el proceso de entrenamiento vocal se puede utilizar una serie de ejercicios que incluyen:

  - Escalas de *legatos* con distintos intervalos hasta llegar a la octava cómoda
  - *Staccatos* con diferentes intervalos para trabajar el control abdominal y el ataque vocal
  - Trabajar con diferentes sonidos facilitadores, como vibrantes para lograr una mayor vibración mucosa
  - Agilizaciones en diferentes tonos
  - Trabajar con una pelota de Pilates , botando al tiempo que se canta para lograr una postura más adecuada y relajada

- Articulación: La articulación son los movimientos de los órganos fonoarticulatorios para transformar el sonido glótico en palabras. De una buena dicción dependerá la calidad del sonido y el logro de un correcto espacio de resonancia. El entrenamiento de la voz hablada debe lograr que el oyente comprenda y oiga correctamente el mensaje oral. Para ello se realizan los siguientes ejercicios:

  - Praxias orofaciales
  - Trabalenguas
  - Agilizaciones de los grupos fonemáticos
  - Práctica áfona y sonora de vocales aisladas combinadas
  - Lectura a diferentes velocidades

- Modulación: Es la manifestación expresiva del mensaje, y la última adquisición del proceso vocal. Para que el discurso sea rico hay que jugar con variaciones de

intensidad, tono, ritmo y pausas, y esto sólo se logra con un correcto entrenamiento de la voz. Dicho esto, el modo en que cada sujeto mezcla y combina estos elementos es personal por lo que depende de quién transmite el discurso.

Algunos de los ejercicios a realizar son:

- Lectura de textos en monodia
- Variación exagerada de frases y textos
- Narración teatralizada
- Decir frases variando puntuación, acentos, pausas y con diferentes emociones

En definitiva, la terapia vocal trata de conseguir la mejor voz posible que permita al paciente comunicarse adecuadamente, y que logre generalizar los mecanismos aprendidos para hacer frente a sus demandas vocales. La terapia vocal es un conjunto de técnicas no quirúrgicas utilizadas para mejorar la calidad vocal, modificar comportamientos y reducir el traumatismo laríngeo [5 Capítulo “Perfeccionamiento vocal” pág. 447-456].

Capítulo 3: Estado de la cuestión

3.1. Aplicaciones de procesado de voz para rehabilitación

Existen en la actualidad un gran número de aplicaciones dedicadas a la rehabilitación de la voz de pacientes adultos así como de niños que sufren distintos problemas de fonación. Al igual que en la aplicación desarrollada en este proyecto, se utilizan imágenes para motivar al usuario en un entorno agradable, lo que hace que se conviertan los ejercicios vocales en un pasatiempo ameno.

Dentro de este tipo de aplicaciones se pueden diferenciar dos tipos. Las que son:

- Controladas por voz
- No controladas por voz

A continuación se explica en qué consiste cada tipo y se citan algunos ejemplos de aplicaciones ya existentes.

3.1.1 Aplicaciones para rehabilitación controladas por voz

Son aplicaciones que utilizan parámetros de la voz como pueden ser el *pitch*, la intensidad, sonidos sordos o sonoros, para controlar sucesos gráficos en la pantalla del dispositivo. La característica principal de este tipo de aplicaciones es que realizan una captura y análisis de la señal de audio que recoge el micrófono. En tiempo real, la interfaz gráfica emite un *feedback* al paciente. Este tipo de aplicaciones son más complejas técnicamente ya que dependen de algoritmos que procesan la señal de audio lo más brevemente posible para dar sensación de respuesta inmediata. Se describen algunas de estas aplicaciones a continuación.

- Sonneta Voice Monitor [9]: Esta aplicación mide el *pitch* y el nivel de presión sonora (SPL). Ha sido diseñada específicamente para medir la voz humana ya sea hablada, cantada, gritada o tarareada. Un medidor y una gráfica en movimiento ofrecen un *feedback* en tiempo real para ejercicios de canto o de terapia de voz. El *pitch* aparece en Hercios o en semitonos y el nivel de presión sonora en decibelios, tal y como se aprecia en la Figura 4.

Figura 4: Ipad e Ipod presentando frecuencia y nivel sonoro en Sonneta Voice Monitor, tomada de [9]

Se permite grabar muestras de voz para reproducir posteriormente o para enviar al terapeuta. Al reproducir una grabación vuelven a visualizarse los parámetros de pitch y niveles de presión sonora. El terapeuta puede grabar ejemplos para el paciente además de fijar unos límites máximos y mínimos para el *pitch* o nivel sonoro, como visualizado en la Figura 5. En cuanto el usuario sobrepasa estos límites, se le notifica trazando la gráfica de color naranja y con una alerta sonora.

Figura 5: Ipad e Ipod mostrando panel de opciones y gráfica naranja, tomada de [9]

Puede ser utilizada con micrófono externo. Para los ejercicios de intensidad de la voz es necesario calibrar el micrófono para la obtención de resultados precisos aunque no imprescindible. Se puede usar la aplicación sin calibración para medir niveles relativos.

- Voxtraining-Airplane [10]: Esta aplicación, creada por una empresa dedicada a la producción de *software* de análisis de señales de voz, trabaja el control de la intensidad de la voz del paciente con un videojuego intuitivo que se inicia con la pantalla de arranque de la Figura 6.

Figura 6: Portada del videojuego Voxtraining-Airplane, tomada de [10]

El jugador controla la trayectoria de vuelo de una avioneta mediante la intensidad de su voz. Cuanto más nivel sonoro más alto vuela la avioneta y viceversa. Además, la captura de nubes permite obtener más puntos.

Figura 7: A la izquierda el videojuego en acción y a la derecha la ventana de calibración, tomada de [10]

Se puede variar el rango dinámico de acuerdo a las necesidades del usuario así como la duración del juego. Antes del comienzo de la partida se requiere un proceso de calibración para mejorar las prestaciones y evitar la influencia del ruido de fondo, tal y como aparece en la Figura 7.

- Voxtraining-Seagull [11]: Esta aplicación está desarrollada por el mismo grupo de investigación que la anterior. Se asemeja en su funcionamiento y lógica cambiando el hecho que el parámetro que se entrena en este caso es el *pitch*. En la Figura 8 se observa la pantalla de arranque.

Figura 8: Portada del videojuego Voxtraining-Seagull, tomada de [11]

El jugador controla el vuelo de una cigüeña dependiendo de la frecuencia de su voz y debe mantener dicha frecuencia en un cierto rango para poder recoger unas pelotas flotantes que le hacen puntuar.

Figura 9: A la izquierda el videojuego en acción y a la derecha la ventana de configuración de pitch, tomada de [11]

En la Figura 9 se ve se puede variar la banda de frecuencias dependiendo de las necesidades del paciente y de si es hombre, mujer o niño.

- Red Light-Green Light [12]: Este juego ha sido desarrollado por la empresa estadounidense Video Voice, que produce juegos para PC usados en la rehabilitación de voces patológicas. Red Light-Green Light consiste en un semáforo que se pone de color verde cuando el ejercicio se realiza correctamente y de color rojo cuando no. Se permite elegir entre tres modalidades para controlar la luz del semáforo: alto o bajo *pitch*, más o menos intensidad, y rápida o lenta velocidad del habla. El terapeuta puede cambiar los rangos dependiendo de lo que necesite practicar el usuario.

Además se muestran mensajes de texto para avisar si es necesario un cambio en la producción de sonido y un gráfico con el rendimiento a lo largo del tiempo jugado tal y como se ve en la Figura 10.

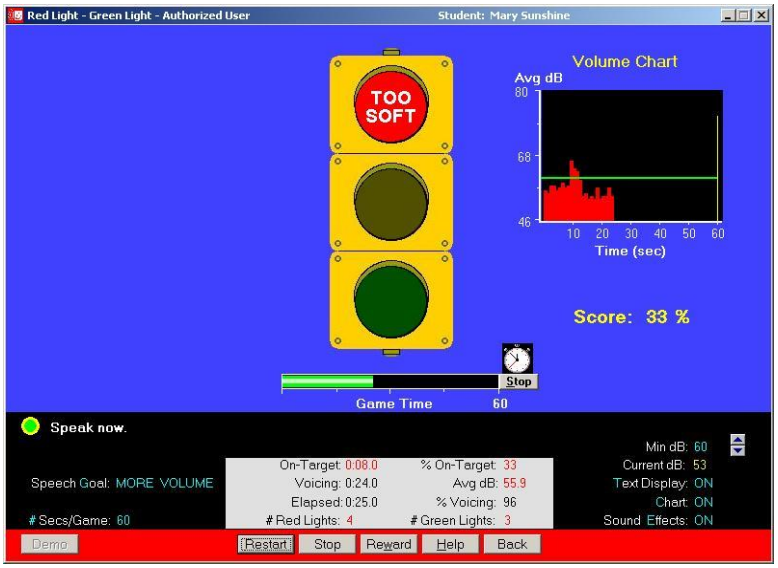

Figura 10: Red Light-Green Light en modo intensidad alta o baja, tomada de [12]

- Speech Therapy 5 [13]: Es un *software* para PC, ha sido desarrollado por una empresa llamada Dr. Speech, que ofrece gran variedad de juegos para la rehabilitación de la voz. Speech Therapy 5 reúne más de 70 juegos para ejercitar en tiempo real cambios en el *pitch*, intensidad, sonidos sonoros y sordos, duración de la fonación, sonorización y detección de vocales. El *software* ofrece un *feedback* colorido e interactivo para los niños. Además se puede grabar y reproducir audio para consultas posteriores. En la Figura 11 se observa a la derecha un juego de *pitch* que consiste en mover la ballena de arriba abajo dependiendo de la frecuencia fundamental y a la izquierda un juego de intensidad que permite desplazar al mono a lo largo de la cuerda dependiendo de la amplitud de la voz del usuario.

Figura 11: Pantallazos de dos juegos incluidos en Speech Therapy 5, tomada de [13]

- KayPENTAX Voice Games [14]: Este *software* también para PC, que reúne un total de diez juegos, ha sido diseñado por la empresa KayPENTAX que vende productos de análisis y rehabilitación de voces patológicas y su *hardware* asociado. Los juegos que se ofrecen permiten al usuario controlar y entrenar el *pitch* de su voz así como la intensidad, la duración de la fonación y el tiempo de sonorización. Se da la opción al usuario de personalizar los juegos para adaptarlos a sus necesidades y añadir dificultad a los ejercicios. En la Figura 12 se aprecia como los juegos tienen distintos entornos gráficos para motivar y amenizar la actividad realizada.

Figura 12: Cuatro de los diez juegos de KayPENTAX Voice Games, tomada de [14]

Las seis aplicaciones que se han comentado anteriormente son desarrolladas para el sistema operativo iOS y para PC. Resulta imposible encontrar aplicaciones de rehabilitación controladas por la voz del usuario para dispositivos Android. De una lista de más de 225 aplicaciones dedicadas exclusivamente a patologías de la voz, ninguna es controlada por la voz [15]. Aun así se ha encontrado un juego para Android llamado SpaceWilli, que no ha sido creado para la rehabilitación de la voz pero que sí utiliza la intensidad de la voz del usuario para controlar el personaje. Éste sube cuanto más elevado sea la intensidad y viceversa. [16]

3.1.2 Aplicaciones para rehabilitación no controladas por voz

Este tipo de aplicaciones enseñan y entrenan a los pacientes a pronunciar correctamente fonemas, palabras o sonidos en general a partir de imágenes, instrucciones o incluso audios. Pretenden ofrecer una herramienta que se asemeje a la labor de un logopeda sin que por ello se prescinda del profesional para el seguimiento de la terapia.

La gran diferencia con las aplicaciones citadas en el apartado anterior es que no se realiza ninguna captura de audio ni por lo tanto procesado de señal. Las mejoras del habla en el paciente dependen del buen uso que este hace de la aplicación y de su compromiso con los ejercicios ya que debe autoevaluarse. Se recomienda que los niños sean supervisados por adultos para su correcta utilización.

- Articulation Station [17]: Esta aplicación para iOS sirve para aprender a pronunciar los sonidos consonantes del habla inglesa con seis actividades distintas. Ha sido creada por terapeutas del habla para ayudar a niños y adultos a practicar palabras clave con actividades divertidas con la supervisión de un especialista, profesor o padre.

Figura 13: Menú de la aplicación con los distintos fonemas a trabajar, tomada de [17]

Cada una de estas actividades tiene como objetivo fonemas de principio, mitad y final de palabras, frases e historias. El usuario debe repetir los sonidos que se le indican. La mayoría de los sonidos (/p/, /b/, /m/, /d/, /n/, /t/, /k/, /g/, /f/, /v/, /ch/, /j/, /z/, /sh/, /th/) que aparecen en la Figura 13, tienen 60 palabras clave para ejercitarse. Se permite grabar voz para una evaluación posterior. En la Figura 14 se visualizan dos de las actividades de la aplicación, una de ellas que consiste en repetir palabras acompañadas de ilustraciones y otra en la que se ejercita un fonema en particular a lo largo de un texto.

Figura 14: Actividades distintas de la aplicación, tomada de [17]

- Speech Companion [18]: Las áreas de aplicación de esta herramienta, desarrollada para Android, incluyen la rehabilitación después de un accidente cerebrovascular o problemas del motor del habla por desequilibrios congénitos en los músculos del área de la boca.

Figura 15: Menú principal y listado de videos demostrativos, tomada de [18]

En la Figura 15 se aprecia el menú principal de SpeechCompanion, así como el listado de videos cortos en el que un logopeda muestra ejercicios motores de boca o lengua. El paciente puede crear listas individuales de entrenamiento con estos videos. Para practicar, la aplicación permite reproducir los videos y mostrar la propia imagen del usuario al hacer uso de la cámara frontal del *smartphone* o tableta. Esto le permite compararse con el terapeuta del habla que aparece en los videos tal y cómo vemos en la Figura 16.

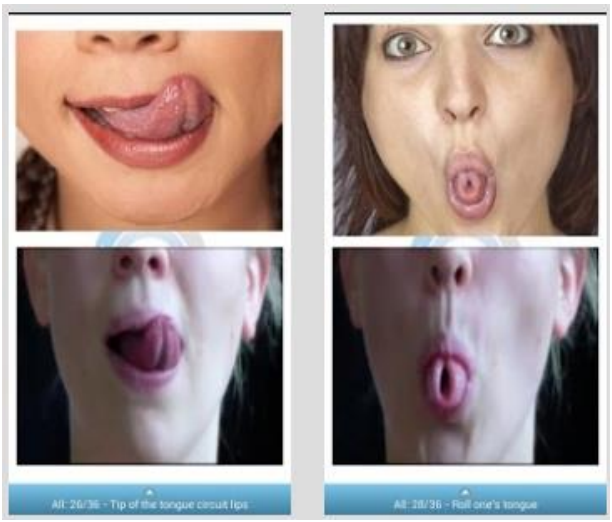

Figura 16: Usuario practicando los ejercicios con la cámara frontal de su dispositivo a la vez que visualiza las imágenes del terapeuta, tomada de [18]

3.2. Herramientas disponibles

Con el auge continuo de la popularidad de Android como sistema operativo de dispositivos móviles, han aparecido un gran número de motores de juego para esta plataforma. Existen algunos para crear videojuegos en 3D, otros en 2D y otros que para ambos. Se diferencian en que algunos son gratuitos y otros no. Algunos ofrecen soporte técnico así como foros y tutoriales para ayudar al desarrollador y se escriben con un cierto lenguaje de programación. En el capítulo siguiente se presenta el motor de juego, Andengine, que se ha utilizado para desarrollar la aplicación, pero antes se exponen los motores cuyo uso ha sido planteado para la elaboración del proyecto.

- Corona SDK [19]:

Figura 17: Logo de Corona SDK, tomado de [19]

La librería API de este motor de juego 2D permite crear desde animaciones a conexiones de red con muy pocas líneas de código. Los cambios realizados en la aplicación pueden ser visualizados instantáneamente gracias al simulador incorporado en el motor. El desarrollo se realiza con Lua, un lenguaje de programación sencillo de aprendizaje. Corona permite publicar aplicaciones para iOS, Android, Kindle Fire y NOOK a partir de un solo código. El motor está basado en OpenGL 2.0 y permite crear efectos cinematográficos con pocas líneas de código. Existen foros en los que otros desarrolladores dan consejos y comparten código para ayudar a los más principiantes. También ofrece guías, tutoriales, vídeos y ejemplos así como herramientas creadas por usuarios para facilitar algunas de las tareas del motor. La versión más básica tiene un coste de 16$ mensuales.

- Cocos2d-x [20]:

Figura 18: Logo de Cocos2d-x, tomado de [20]

Es un motor de juego de código abierto que puede ser usado para crear juegos, aplicaciones y otras GUI multiplataforma. Basado en OpenGL 2.0, usa los lenguajes

de programación Lua, C++ y Javascript para su desarrollo en las plataformas iOS, Android, Windows Phone, Mac OS X, Windows Desktop y Linux. Es un motor veloz con poderosas características. Lo usan grandes compañías de desarrollo así como profesionales independientes. Es gratuito pero no ofrece tantos tutoriales y ayudas como otros motores.

- Libgdx [21]:

Figura 19: Logo de Libgdx, tomado de [21]

Es un *framework* libre y gratuito para el desarrollo de videojuegos escrito en Java con sus partes más críticas implementadas en C/C++. Corre sobre OpenGL 1.0 y 2.0 para dispositivos actuales. Tiene módulos para manejar gráficos, audio y entrada del usuario de una manera sencilla. Soporta las plataformas Windows, Linux, Android, Mac OS X, Javascript/WebGL (GWT) e iOS aunque en este último caso necesita Monotouch cuya licencia no es gratuita ni barata. Se pueden encontrar una cantidad razonable de tutoriales y ayudas para desarrollar los juegos.

- Marmalade SDK [22]:

Figura 20: Logo de Marmalade SDK, tomado de [22]

Como su nombre indica es un kit de desarrollo de *software* y un motor de juego multiplataforma que contiene librerías de ficheros, ejemplos, documentación y herramientas para desarrollar, probar y desplegar aplicaciones para dispositivos móviles. La versión más económica tiene un coste de 15$ mensuales y sólo permite desarrollar para iOS y Android. Este motor también basado en OpenGL 1.0 y 2.0, utiliza el lenguaje Lua de programación para la creación de sus juegos y ha sido usado para crear un buen número de juegos móviles conocidos hoy en día.

3.3. Comparativa de herramientas

Las características que más van a interesar para la elección del motor de juego serán básicamente tres:

- Coste del software
- Calidad del soporte técnico disponible
- Lenguaje de programación utilizado

A continuación se presentan estas características en una tabla comparativa:

Motor de juego	Precio	Soporte técnico	Lenguaje de programación
Corona SDK	16$ mensuales	Bueno	Lua
Cocos2d-x	gratuito	Bueno	Lua
Libgdx	gratuito	Muy bueno	Java
Marmalade	15$ mensuales	Escaso	Lua
Andengine	gratuito	Muy bueno	Java

Tabla 1: Tabla comparativa de los motores de juego

Para el desarrollo de este proyecto se dispone de una financiación específica por lo que los motores de juego que no son gratuitos son descartados. Además se da importancia al soporte técnico que cada uno ofrece, desde la información ofrecida en las webs oficiales, los tutoriales de particulares o de los creadores de los motores, los foros de discusión hasta los libros dedicados específicamente a ellos. Finalmente, se ha dado importancia al lenguaje de programación que deberá usar el desarrollador para crear la aplicación.

Ya que el autor de este proyecto está más familiarizado con el lenguaje de programación Java que con el lenguaje Lua, los motores que ofrecen el primero verán su posibilidad de elección aumentada. Además, ya que el sistema de captura de muestras de audio del micrófono se realiza con la clase AudioRecord para aplicaciones Java, la integración del motor de juego y dicho sistema de captura se hace más sencillo utilizando este lenguaje únicamente.

Debido a las características prioritarias mencionadas anteriormente, se desechan en primer lugar los motores Corona SDK y Marmalade, principalmente por el coste económico que suponen. También es desechado el motor Cocos2d-x ya que usa Lua como lenguaje de programación. Libgdx y Andengine son motores que ofrecen ventajas muy similares a la hora de desarrollar videojuegos para plataformas móviles, en este caso Android, y que podrían servir para la creación de la aplicación detallada en este trabajo.

El hecho de que sea gratuito, ofrezca gran cantidad de tutoriales y ayudas en la red,

utilice lenguaje Java, haber sido concebido para Android y, sobretodo, que sea el que se integre de forma más sencilla al entorno de esta plataforma, han sido las razones por las que se ha elegido Andengine como motor de desarrollo para este proyecto frente a los demás.

Figura 21: Logo de Andengine, tomado de [23]

Capítulo 4: Descripción de la solución propuesta

4.1. Selección de herramientas

En este apartado de la memoria vamos a comentar las distintas herramientas usadas para desarrollar la aplicación.

4.1.1 ADT y Eclipse

La utilización de Eclipse con el *plugin* ADT es la configuración más popular y recomendada del entorno de desarrollo por los desarrolladores de Android.

El *plugin* ADT para Eclipse integra muchas de las herramientas más importantes de Android SDK en su entorno de desarrollo, y proporciona varios asistentes para crear, depurar e implementar aplicaciones Android. El *plugin* ADT añade un conjunto de útiles funciones al IDE (Integrated Development Environment) de Eclipse para la creación rápida de prototipos que a continuación pueden ser probadas.

El editor de lenguaje de programación Java contiene características IDE comunes, tales como la comprobación de la compilación de sintaxis, el auto-completado, y la documentación integrada para las API de Android. ADT también proporciona editores XML personalizados que permiten editar archivos XML específicos de Android. Un editor gráfico permite el diseño de interfaces de usuario con una interfaz de arrastrar y soltar [24].

Eclipse organiza su espacio de trabajo en perspectivas (cada una formada por un conjunto de paneles determinados) para realizar diferentes tareas como escribir código o depurar. Puede alternar entre perspectivas seleccionando la pestaña adecuada en la parte superior derecha del entorno Eclipse. La perspectiva DDMS integra la herramienta DDMS (Dalvik Debug Monitor Service, Servicio de monitorización de depuración Dalvik) en Eclipse de forma que se pueda agregar el emulador e instancias de dispositivos para depurar las aplicaciones. En la Figura 22 se muestra la barra de herramientas de Eclipse con las características añadidas de Android [25].

Figura 22: Características de Android añadidas a la barra de herramientas y perspectivas en Eclipse

Para probar y depurar aplicaciones en Android se puede usar o bien un dispositivo real o un emulador. El emulador de Android es una de las herramientas más importantes de Android SDK. Se utiliza con frecuencia para diseñar y desarrollar aplicaciones. El emulador se ejecuta en el ordenador y se comporta tal y como lo haría un dispositivo móvil. Se puede apreciar esto en la Figura 23. Es un dispositivo genérico y no está vinculado a ninguna configuración de teléfono específica.

Figura 23: Emulador de Android en PC

También se puede probar y depurar las aplicaciones directamente en un dispositivo real de Android. Sólo hace falta configurar el sistema operativo para tener acceso a través del cable USB, como se puede ver en la Figura 24 siguiente.

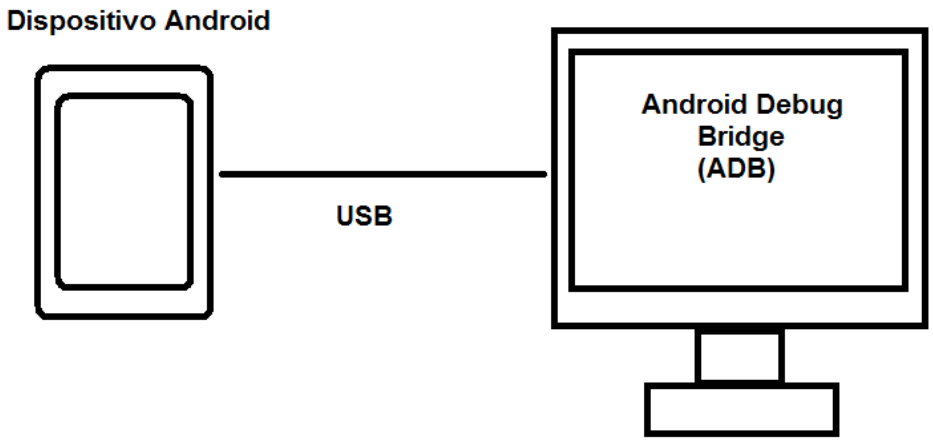

Figura 24: Dispositivo Android conectado a través de USB a un ordenador para depurar

4.1.2 Motor de juego, Andengine

AndEngine es un *framework* 2D de código abierto desarrollado por Nicolas Gramlich, completamente equipado y gratuito para la plataforma Android. Está basado en la API multilenguaje y multiplataforma OpenGL y usa puramente el lenguaje Java. Es uno de los pocos motores de juego en 2D para la plataforma Android, que está siendo usado constantemente para crear juegos con estilo y divertidos por desarrolladores independientes y profesionales por igual. Incluso se ha utilizado en algunos de los juegos de mayor éxito que están en el mercado hasta la fecha [26, Capítulo "Preface", pág. 1].

El orden de las operaciones es importante cuando se trata de la inicialización de un juego. Las necesidades básicas de un juego incluyen: crear el motor, cargar los recursos del juego, crear la pantalla inicial y la configuración. Esto es todo lo que se necesita para las bases de un juego AndEngine. Sin embargo, si queremos algo más complejo, es conveniente conocer el ciclo de vida completo de un juego en AndEngine.

A continuación, se cubren los métodos de ciclo de vida en el orden en que son llamados desde la puesta en marcha de una actividad hasta que se termina.

Las llamadas de ciclo de vida del juego durante su lanzamiento son las siguientes:

- `onCreate`: Este método es el punto de entrada de la aplicación nativa del SDK de Android. En AndEngine, este método simplemente llama al método `onCreateEngineOptions( )` en la clase `BaseGameActivity` para luego aplicar las opciones devueltas al motor de juego.

- `onCreateGame`: Sirve para manejar el orden de ejecución de las tres siguientes llamadas del ciclo de vida AndEngine.

- `onCreateResources`: Este método permite declarar y definir los recursos iniciales de la aplicación que son necesarios durante el lanzamiento de la actividad. Estos recursos incluyen, texturas, sonidos y música, así como las fuentes.

- `onCreateScene`: Aquí, se hace cargo de la inicialización de la escena de la actividad. Es posible unir objetos a la escena dentro de este método, pero para mantener las cosas organizadas, por lo general es mejor adjuntar los objetos dentro de `onPopulateScene( )`.

- `onPopulateScene` : El método `onPopuplateScene( )` se utiliza para definir el resultado visual de la escena cuando la aplicación se inicia por primera vez. Hay que tener en cuenta que la escena ya está creada y aplicada al motor en este punto.

- `onResumeGame` : Aquí se tiene la última llamada método que se lleva a cabo durante el ciclo de arranque de una actividad. Si la actividad llega a este punto sin ningún problema, se llama al método `start( )`.

Las llamadas de ciclo de vida del juego durante su minimización o finalización son las siguientes:

- `onPauseGame`: Este método llama al método `stop( )` del motor, causando la paralización de todos los controladores e hilos de actualización del motor.

- `onGameDestroyed`: Por último, se llega a la última llamada del ciclo de vida completo de AndEngine. Se establece una variable `mGameCreated` booleana que indica que la actividad ya no está en funcionamiento [26, Capítulo "Andengine Game Structure", pág. 10-12].

En la Figura 25, se puede ver cómo sería el ciclo de vida en acción, al crear, minimizar y destruir el juego.

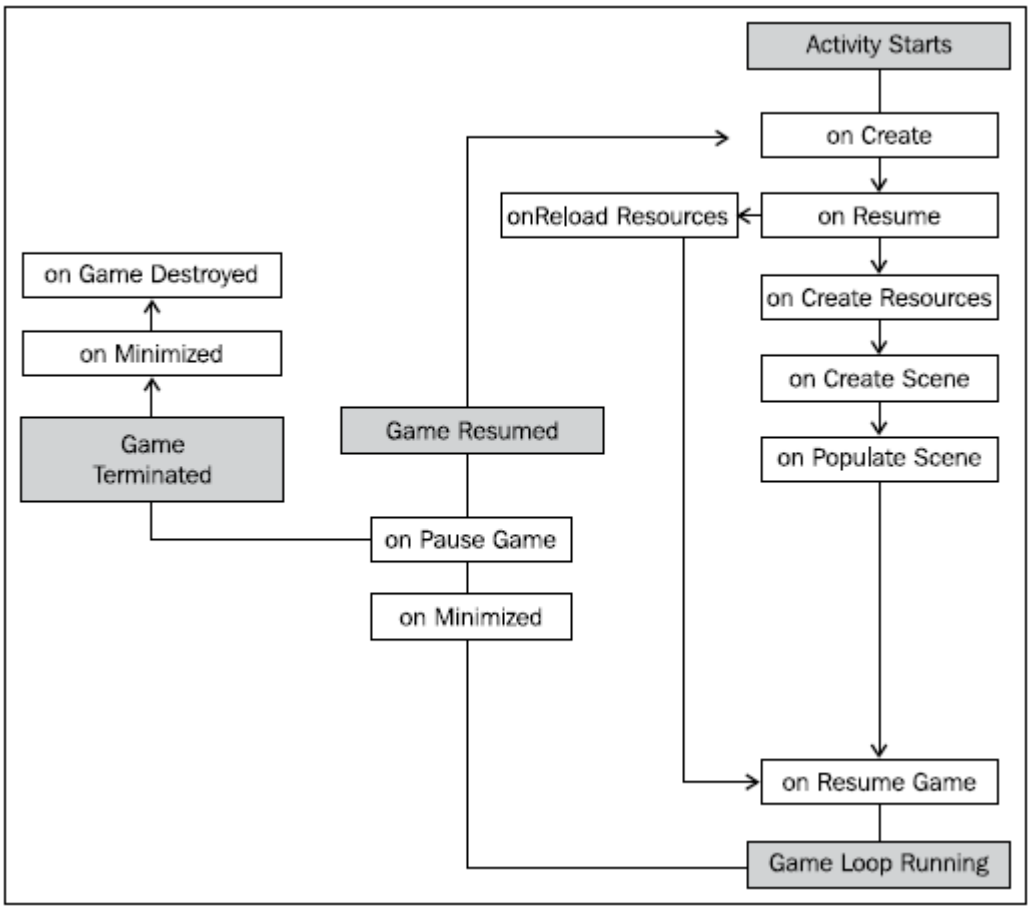

Figura 25: Diagrama de bloques mostrando el ciclo de vida habitual de un juego en Andengine [26, Capítulo "Andengine Game Structure", pág. 12]

4.1.3 Librería de FFT, JTransforms

JTransforms es la primera librería FFT escrita en puro Java que es de código abierto y multihilo. Se usa esta librería en aplicaciones musicales, de visualización del audio en tiempo y frecuencia o bien de reconocimiento de patrones. En la actualidad esta librería permite realizar cuatro tipos de transformadas distintas:

- Transformada discreta de Fourier, DFT
- Transformada discreta del coseno, DCT
- Transformada discreta del seno, DST
- Transformada discreta de Hartley, DHT

Características:

- Es la implementación más veloz de DFT, DCT, DST y DHT en Java puro
- Realiza transformadas de una, dos y tres dimensiones
- Los datos pueden tener un tamaño arbitrario
- Precisión simple y doble
- Variantes unidimensionales y multidimensionales de transformadas en 2D y 3D
- Multihilado automático
- FFT optimizadas para datos de entrada de tipo real (40% más rápido que los de tipo complejo) [27]

4.2. Estructura general del software desarrollado

4.2.1. Descripción general del software desarrollado

La aplicación desarrollada en este trabajo es un proyecto de Java para Android llamado SplashProject, compuesto por dos actividades enlazadas entre sí y al que se le han adjuntado la librería Andengine del motor de juego y la librería JTransforms para el procesado de audio.

La actividad principal se denomina SplashActivity y en ella se encuentran los

métodos principales para el funcionamiento de la aplicación así como las llamadas a funciones presentes en la segunda actividad, llamada SceneManager, que sirve para administrar cada una de las escenas del videojuego.

Esta aplicación se inicializa con una pantalla de arranque, o *splash screen* en inglés, cuya utilidad es la de hacer ver al usuario que se está ejecutando el programa mientras aparece el logo de la aplicación y se cargan los distintos recursos que se utilizarán en el menú y en los juegos. Esto se hace para que no exista demora al pasar de una escena a otra debido al tiempo de carga de los recursos. Es preferible cargar todo lo necesario al principio para luego simplemente disponer de ello en cuanto haga falta. Una vez desaparece dicha pantalla de arranque el usuario se encuentra con el menú. En ésta pantalla puede seleccionar entre los tres juegos propuestos el que desee ejecutar. Estos juegos permiten entrenar:

- El *pitch*
- La intensidad
- Los sonidos sordos y sonoros

En el diagrama de bloques de la Figura 26 se aprecian las distintas escenas existentes en la aplicación.

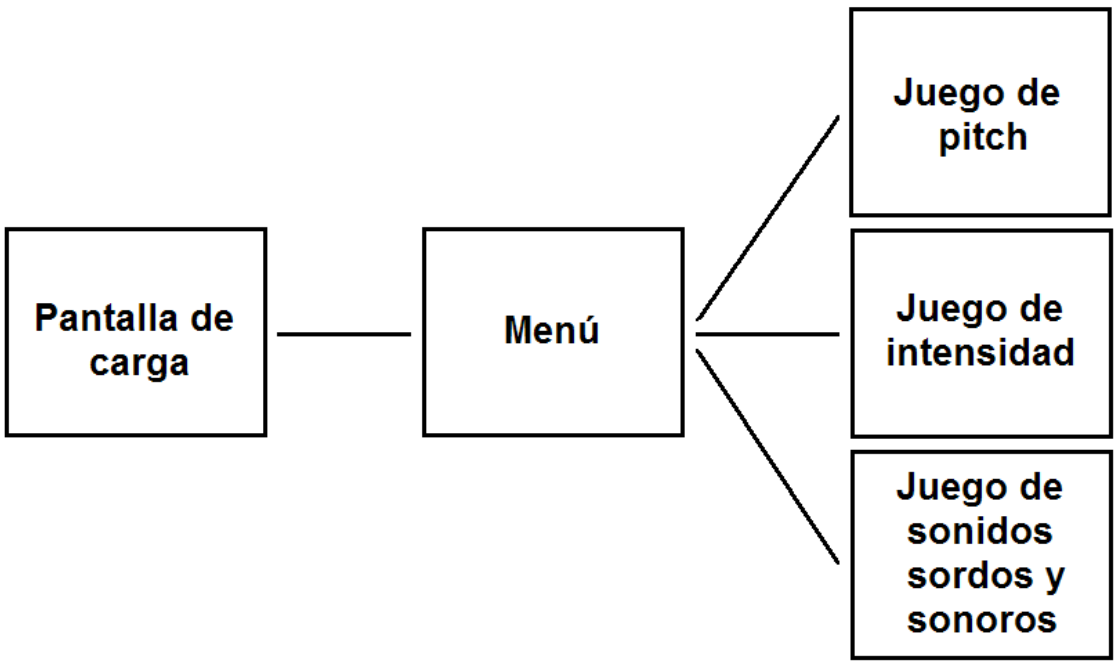

Figura 26: Diagrama de bloques del orden de las escenas de la aplicación

Para cada uno de los juegos se ha desarrollado una interfaz gráfica distinta y unos algoritmos de audio específicos. Se describe brevemente a continuación el

funcionamiento de los tres juegos antes de pasar al siguiente apartado para estudiar el software desarrollado paso a paso.

- **Juego de pitch**

  El juego consiste en un helicóptero cuya posición vertical en pantalla depende de la frecuencia fundamental de la voz del usuario. Cuanto más alta la frecuencia más alto se posiciona el helicóptero y viceversa. Para la detección de *pitch* se ha elegido el método cepstrum y se puede visualizar la frecuencia gracias a un contador numérico. Además se ha añadido un fondo deslizante al juego para crear una sensación de movimiento del helicóptero con respecto a las demás imágenes.

- **Juego de sonidos sonoros y sordos**

  En este juego dependiendo del sonido producido por el usuario se visualiza una imagen distinta en pantalla. Si el sonido detectado tiene un nivel de intensidad inferior a un cierto umbral aparece una primera imagen, si el nivel del sonido supera el umbral y además es sordo aparece una segunda imagen y si el nivel del sonido supera el umbral y es sonoro aparece una tercera imagen. Para la diferenciación de sonidos sonoros y sordos se ha optado por el cálculo de energía logarítmica así como el método de cruces por cero.

- **Juego de intensidad**

  Este juego se asemeja al anterior pero el parámetro que interesa especialmente es la intensidad del sonido producido por la voz del usuario. Dependiendo de si se detecta un nivel de la señal de audio por encima o debajo de un cierto umbral se muestra en pantalla una imagen u otra. Para obtener la intensidad se calcula la energía logarítmica.

4.2.2. Descripción detallada del software y de la interfaz de usuario

Tal y como se ha visto en el apartado anterior, la aplicación se compone de una actividad principal, SplashActivity, en la que se definen los cuatro métodos básicos incluidos en el ciclo de vida de un juego en Andengine además de una función adicional que permite desplazarse de escena al pulsar el botón de atrás del dispositivo Android que se esté utilizando. Estas cinco funciones que se encuentran en la actividad son:

- `onCreateEngineOptions`
- `onCreateResources`
- `onCreateScene`

- `onPopulateScene`

- `onBackPressed`

Por otro lado, para que la actividad no sea un caos y no desarrollar la totalidad del juego en una sola clase, se crea un administrador de escenas. Hay tantas escenas como pantallas distintas dentro de la aplicación, en este caso se han desarrollado cinco escenas (pantalla de carga, menú, juego de pitch, juego de intensidad y juego de sonidos sonoros y sordos). Sin mencionar todavía los nombres exactos de cada una de las funciones que componen esta actividad denominada SceneManager, se puede decir que permite:

- Crear un listado de las escenas que permite manejar todas las escenas del juego

- Cargar los recursos de la pantalla de arranque y visualizar dicha pantalla de arranque mientras tanto

- Establecer todas las escenas y cargar sus respectivos recursos

- Establecer un método que permite pasar de una escena a otra

- Establecer un método que devuelva la escena actual

A partir de este momento se describe el *software* realizado paso a paso, comentando los varios métodos y funciones utilizados en ambas actividades.

Dentro de la clase SplashActivity creada, que extiende a su vez la clase BaseGameActivity de Andengine, se encuentran los cuatro métodos básicos para el desarrollo del videojuego. Cada uno de estos métodos, salvo el primero `onCreateEngineOptions`, ofrece una retrollamada o *callback* en inglés. El propósito de este *callback* es de avisar al programa de que se ha llegado al final del método en cuestión y que se puede pasar al siguiente. Esto supone una cierta flexibilidad que resulta beneficiosa a la hora de cargar el juego.

- **`onCreateEngineOptions`**

  El primer método, `onCreateEngineOptions`, sirve, tal y como su nombre indica, para crear las opciones del motor de juego y devolverlas al final de dicho método. El motor de juego sirve para determinar la apariencia del mismo.

  En primer lugar se define la cámara, denominada `mCamera`, así como su altura y anchura. Se elige como tamaño 800 píxeles de ancho y 480 píxeles de alto que son las dimensiones típicas para aplicaciones desarrolladas con el motor Andengine.

  A continuación se definen los cuatro parámetros de las opciones del motor. El primer parámetro establece que el videojuego aparezca a pantalla completa en el

dispositivo en el que se visualizará. Como segundo parámetro se elige que la orientación de la pantalla sea en modo panorámico. El `RatioResolutionPolicy`, cuya utilidad es evitar deformaciones en las imágenes independientemente del dispositivo utilizado para ejecutar el juego, es el tercer parámetro. Para ello se le pasan las dimensiones anteriormente citadas de la cámara. Finalmente se define `mCamera` como la cámara que se quiere usar para el motor de juego.

- **`onCreateResources`**

Después de establecer las opciones del motor, se llama al método `onCreateResources`. En él se inicializa el objeto sceneManager que permite acceder a las funciones implementadas en la actividad SceneManager y ocuparse de la carga y creación de los distintos recursos y escenarios de la aplicación. En vez de cargar en este momento todos los recursos necesarios, sceneManager se encarga de cargar simplemente los recursos necesarios para la pantalla de arranque. De esta forma el usuario podrá visualizar rápidamente algo en pantalla sin tener la sensación de que el juego se ha congelado. Se realiza por lo tanto una llamada a la función `loadSplashResources` de la actividad SceneManager.

  - `loadSplashResources`

    Esta función carga la imagen de la Figura 27 durante la pantalla de arranque y que en este caso es el logo de la aplicación.

Figura 27: Logo de la aplicación

Para ello hay que especificar en qué carpeta se encuentra la imagen. Dentro del proyecto creado se sigue la ruta `assets/gfx` para ubicar dicha imagen. De aquí en adelante siempre se hará referencia a este directorio para los gráficos utilizados en esta aplicación. Lo siguiente es definir las dimensiones de la imagen para a continuación precisar el nombre del archivo de imagen requerido y finalmente cargarla. Una vez la función `loadSplashResources` ha finalizado y

ya de vuelta en el método `onCreateResources` se hace uso del *callback* para poder desplazarse al siguiente método `onCreateScene`.

- **`onCreateScene`**

Ahora que ya se han cargado los recursos necesarios para la pantalla de arranque se puede crear dicha pantalla con sceneManager. El método `onCreateScene` tiene una estructura muy sencilla ya que simplemente hace uso del *callback* para llamar a la función `onCreateSplashScene` implementada en la actividad SceneManager.

  - `onCreateSplashScene`

    En esta función se crea una escena nueva y se define el color del fondo de pantalla a negro. A continuación se crea un *sprite* que será la imagen del logo de la aplicación anteriormente visto. Se establecen sus coordenadas para que aparezca en el centro de la pantalla. Además se le aplica a la imagen la técnica de *dithering* para que el resultado visual sea más uniforme y suave. Finalmente se amarra el *sprite* a la escena y se devuelve la escena completa. En este preciso momento la pantalla de arranque se hace visible. El resultado final queda tal y como se puede ver en la Figura 28.

Figura 28: Pantalla de arranque o splash screen

- **`onPopulateScene`**

En el siguiente método implementado, `onPopulateScene`, se cargan todos los recursos necesarios para los juegos mientras se visualiza la pantalla de arranque para finalmente crear y mostrar la escena posterior que será el menú. Para ello se utiliza un `TimeHandler` que permite hacer aparecer la pantalla de arranque durante tres

segundos, que es la duración que se ha elegido. Mientras tanto se cargan los recursos del menú y de los tres juegos desde la actividad SceneManager.

- `loadMenuResources`

  La función que se encarga de los recursos del menú es `loadMenuResources`. En esta función se carga la imagen de la Figura 29 que servirá de fondo para el menú así como tres imágenes que servirán de botones para la selección de los juegos. Se define la ruta del directorio en la que están las imágenes para luego establecer las dimensiones del `TextureAtlas` en la que se dispondrán dichas imágenes. Se precisa el nombre de los archivos imagen del fondo y los botones.

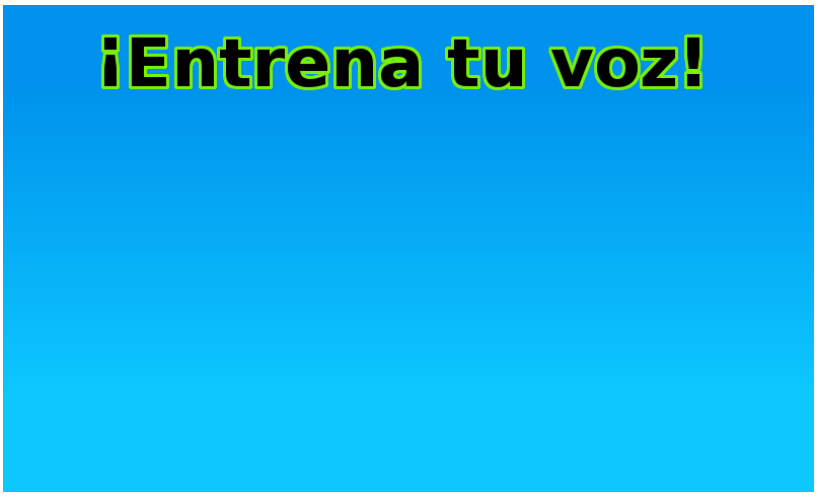

Figura 29: Fondo del menú

Para las imágenes de los botones se especifica que están divididas en dos partes. La parte izquierda de la imagen se muestra cuando el botón no ha sido pulsado y la imagen de la derecha, con más luminosidad, indica que ha sido pulsado. Esto se puede apreciar a continuación para los tres botones de los juegos en la Figura 30.

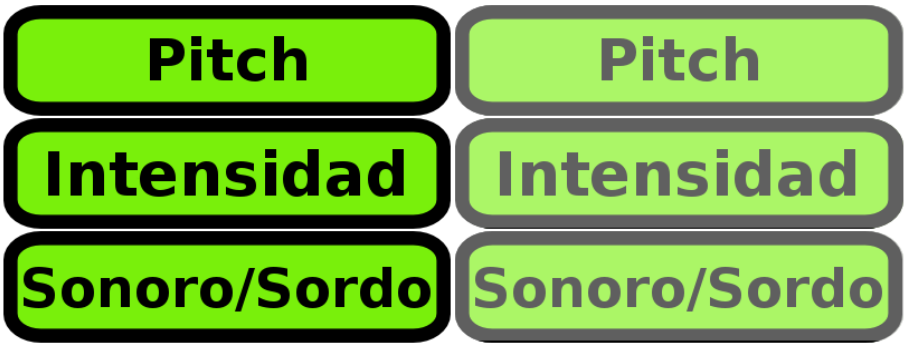

Figura 30: Botones tipo mosaico para los tres juegos

- `loadGame1Resources`

La siguiente función que es llamada mientras aparece la pantalla de arranque es `loadGame1Resources` que corresponde a la carga de recursos del juego de pitch. En dicho juego aparece un helicóptero que sube o baja dependiendo de la frecuencia detectada a partir de la señal de voz capturada por el micrófono.

En esta escena además se tiene un contador que muestra el valor numérico de la frecuencia en cuestión.

Para dar sensación de movimiento al helicóptero se usa la técnica de *parallax scrolling* o en castellano, desplazamiento diferencial. Esta técnica consiste en que las imágenes de fondo se desplacen más lentas que las que están situadas en los primeros planos con respecto a la cámara. Esto crea una sensación de profundidad en imágenes 2D para videojuegos. En este juego se deciden utilizar tres fondos distintos.

En `loadGame1Resources` se define en primer lugar el tipo de letra con el que se va a escribir la frecuencia que aparecerá en pantalla. Se elige el tipo de fuente, estilo de letra, tamaño y color. En este caso se opta por una letra en negrita, con color amarillo y tamaño suficiente para que sobresalga en la escena.

Después de haber cargado los parámetros del contador se definen las imágenes correspondientes al helicóptero y fondos del juego. Como ya viene siendo habitual se precisa la ubicación, tamaño y nombre de dichos archivos de imagen. El helicóptero será animado por lo que su imagen se compone por dos *frames* que le darán esa apariencia de movimiento al pasar de uno a otro rápidamente. Se puede visualizar la imagen del helicóptero con sus dos *frames* en la Figura 31.

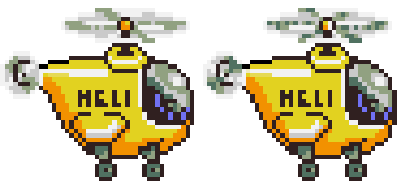

Figura 31: Imagen del helicóptero con sus dos frames, tomada de [28]

Seguidamente se definen también las tres imágenes que formarán el fondo del juego. Dos de ellas son usadas para el *parallax scrolling*. La primera, en la Figura 32, representa una nube que se irá desplazando lentamente por el cielo y la segunda, en la Figura 33, unos montículos sobre los que crecen unos cactus.

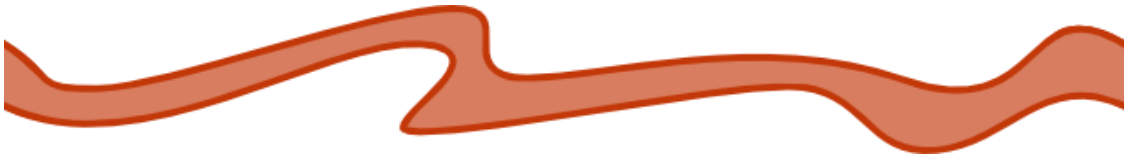

Figura 32: Imagen de nube, tomada de [28]

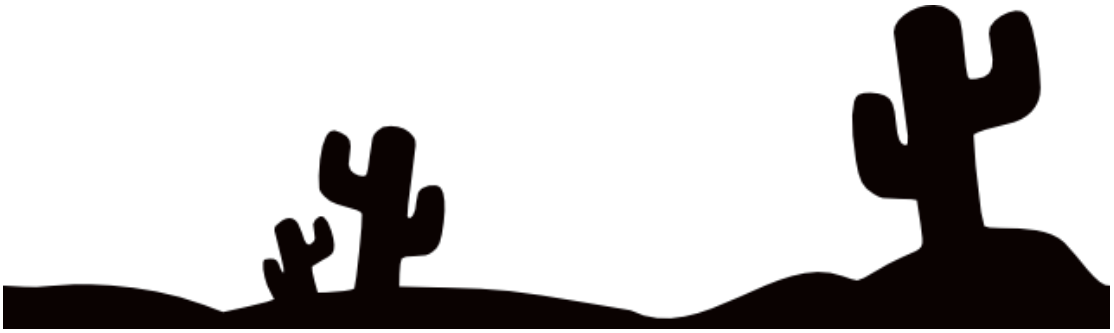

Figura 33: Imagen de montículos sobre los que crecen unos cactus, tomada de [28]

La tercera imagen, en la Figura 34, que sirve de fondo será estática y presenta un paisaje desértico.

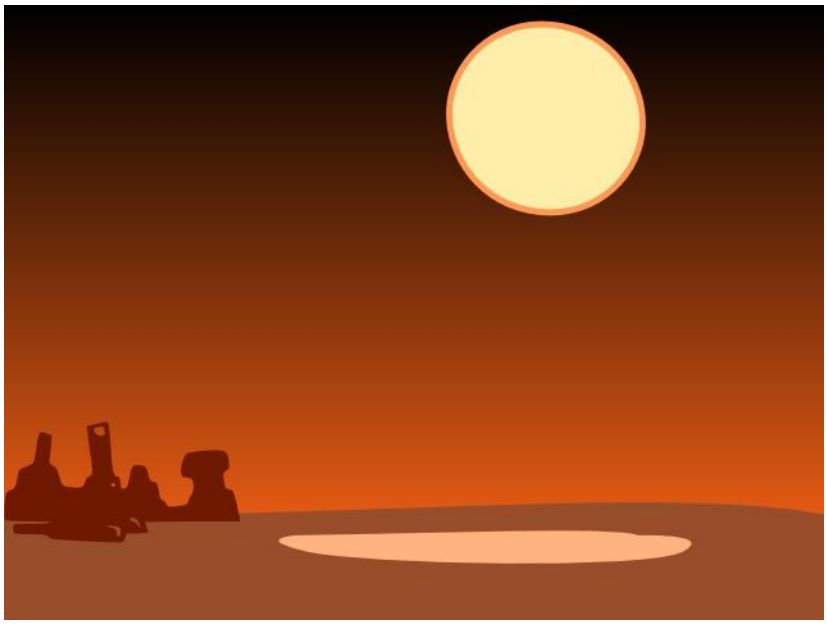

Figura 34: Imagen de fondo estática del juego de pitch, tomada de [28]

Después de cargar las tres imágenes se crean los parámetros de audio que serán los mismos para los tres juegos. En primer lugar se crea el *buffer* de datos en el que se guardarán las muestras de audio que son recogidas por el micrófono.

Seguidamente se crea el objeto AudioRecord. Para ello, hace falta especificar la fuente de audio con la que se recoge la señal. En este caso es el micrófono integrado en el dispositivo móvil. Además hay que precisar la frecuencia de muestreo con la que se toman las muestras de audio, la configuración del canal de audio, el formato en el que se presentan los datos de audio y el tamaño del *buffer* en el que se escribirán los datos de audio durante la grabación.

- `loadGame2Resources`

La siguiente función llamada desde el método `onPopulateScene` mientras aparece la pantalla de arranque es `loadGame2Resources` que se ocupa de cargar los datos del juego de sonidos sonoros y sordos.

Una vez más, se define el directorio en el que encuentran las imágenes que usará el juego así como sus dimensiones. Dependiendo de si la señal de audio grabada por el micrófono tiene un nivel inferior a un cierto umbral, tiene un nivel superior a dicho umbral y es sordo, o bien superior al umbral y sonoro, aparece en pantalla una imagen diferente. Por esta razón la imagen tipo mosaico usada está dividida en tres partes tal y como se aprecia en la Figura 35.

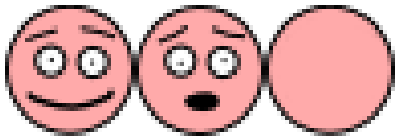

Figura 35: Imagen tipo mosaico del juego de sonidos sordos y sonoros con tres estados diferentes, tomada de [28]

De la misma forma que se ha hecho para el juego descrito anteriormente también en este se crea un *buffer* en el que guardarán las muestras de audio y el objeto AudioRecord que permite establecer distintos parámetros para la grabación.

- `loadGame3Resources`

Esta función loadGame3Resources carga los datos del juego cuyo objetivo es ejercitar la intensidad de la voz del usuario. La estructura de dicha función es idéntica a la anterior salvo por la imagen utilizada.

En este juego dependiendo de si se supera un umbral de intensidad determinado, la imagen que aparece en pantalla variará de un estado a otro para avisar al usuario que ha superado dicho umbral. En la Figura 36 se ve como la imagen de tipo mosaico tiene dos partes.

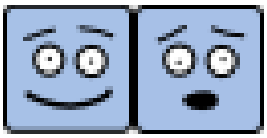

Figura 36: Imagen del juego de intensidad con dos estados diferentes, tomada de [28]

Tal y como se ha hecho en los dos juegos anteriores, se crea un *buffer* y el objeto AudioRecord para la grabación de audio.

- `createMenuScene`

Después de haber cargado todos los recursos necesarios para la aplicación y mientras se sigue mostrando la pantalla de arranque, se crea la siguiente escena de la aplicación que es el menú. En él aparece el fondo y los tres botones que se cargaron anteriormente en `loadMenuScene`. Estos botones permiten acceder a cada uno de los tres juegos tal y como se verá seguidamente.

Lo primero que se hace en la función `createMenuScene` es crear una escena nueva. Se define la imagen que hace de fondo en el menú como un *sprite*. Se especifica que las dimensiones de dicho *sprite* son las mismas que las de la anchura y altura de la cámara que se definió al crear las opciones del motor de juego, para que ocupe toda la pantalla del dispositivo. Se le aplica a esta imagen el efecto *dithering* para que quede más suave y uniforme.

A continuación se crean e instalan los botones que permiten acceder a los tres juegos desarrollados. Bajo el título de la aplicación se posicionan los tres botones del juego siendo el primero de ellos el que permite seleccionar el juego de pitch, el del medio seleccionar el juego de sonidos sonoros y sordos y el de abajo del todo el juego de la intensidad de la voz (Figura 37).

Figura 37: Pantalla del menú principal

Gracias al comando `onAreaTouch` se consigue, al pulsar con el dedo encima del área de uno de los botones que aparecen en la pantalla del dispositivo móvil, que la imagen de dicho botón cambie a la de la imagen del botón, vista anteriormente en `loadMenuResources`, que tiene mucha luminosidad. Esto sirve de *feedback*, para que el usuario se dé cuenta que ha pulsado el botón correctamente tal y como se ve en la Figura 38.

Figura 38: Botón pulsado del juego de sonidos sonoros y sordos

Cada vez que se pulsa uno de los tres botones se crea el juego que se ha seleccionado con las funciones `createGame1Scene`, `createGame2Scene` o `createGame3Scene` y se cambia de escena con la función `setCurrentScene` que se comenta brevemente a continuación.

- `setCurrentScene`

Antes de comentar el funcionamiento de `setCurrentScene`, en la actividad SceneManager se crea un `enum` llamado `AllScenes` en el que se definen las distintas escenas que tiene la aplicación. En este caso se cuenta como ya se sabe con cinco escenas: la de la pantalla de arranque llamada `SPLASH`, la del menú llamada `MENU`, la del juego de pitch llamada `GAME1`, la del juego de intensidad llamada `GAME2` y la del juego de sonidos sonoros y sordos llamada `GAME3`.

Para poder desplazarse a una escena se usa un *setter* que es un método de acceso que permite visualizar una escena en pantalla. Esta función es en este caso `setCurrentScene`. Con los comandos `switch` y `case` se establece una escena u otra.

- `createGame1Scene`

En cuanto se pulsa el botón del juego de pitch en el menú de la aplicación, se llama a la función `createGame1Scene`. Esta función permite crear una nueva

escena así como colocar los distintos elementos que se han cargado anteriormente en la función `loadGame1Resources` según se precise. Además es aquí donde se inicializa la grabación de audio para a posteriori llamar a la función que permitirá calcular la frecuencia fundamental de la voz del usuario.

En primer lugar se crea una nueva escena para el juego de pitch. A continuación se crea el contador en el que se muestra el valor de la frecuencia fundamental. Hay que precisar las coordenadas en las que se posiciona dicho contador así como el tipo de letra que usa para escribir los números y la cantidad máxima de dígitos que se pueden escribir. En este caso son tres ya que la frecuencia fundamental de la voz del usuario no superará los 999 Hz.

Después de anclar el contador a la escena se desea posicionar el *sprite* del helicóptero. Al ser el objeto principal del juego se le ubica inicialmente en el centro de la pantalla. Se le define como un *sprite* animado y se precisa la velocidad a la que el motor cambia de un *frame* de la imagen del helicóptero a otra. Esto hace que se cree una sensación de que las hélices del helicóptero van girando a gran velocidad y por lo tanto de movimiento. Además se multiplica por dos las dimensiones del *sprite* del helicóptero ya que quedaría demasiado pequeño en la escena. Finalmente se ancla el *sprite* a la escena.

Las siguientes imágenes que se añaden son las que componen el fondo y que sirven para dar profundidad a la escena. La primera es la que sirve de fondo y que es estática. Se posiciona de tal forma que ocupe toda la pantalla del dispositivo. Al ser una imagen muy visible se le aplica el efecto *dithering* para que aparezca más suave a los ojos del usuario. En segundo lugar se posiciona a cierta altura en la escena la imagen de la nube que atraviesa el cielo. En tercer y último lugar se coloca en la parte inferior de la pantalla la imagen de los cactus y montículos de arena.

El paso siguiente consiste en especificar la velocidad y el sentido en el que se desplazan estas tres imágenes. Cómo ya se ha mencionado anteriormente, la imagen de fondo es estática por lo que no se le asigna velocidad alguna. Para dar esa sensación de movimiento deseada, gracias a la técnica del *parallax scrolling*, se le da mayor velocidad a los elementos que se encuentran más cercanos al primer plano. En este caso, a los cactus y montículos que se mueven hacia la izquierda. La velocidad de la imagen de la nube es por lo tanto inferior, quedando en segundo plano.

Lo que viene a continuación es sin duda la parte más importante del juego, en la que se mueve el helicóptero variando sus coordenadas verticales dependiendo de la frecuencia fundamental detectada por la función `frequency`. Esto se produce gracias a un *update handler*, que maneja las actualizaciones de la escena, cada vez que un nuevo *frame* es creado por el motor de juego.

Dentro del *update handler* lo primero que se realiza es una llamada a la

función `frequency` cuyo funcionamiento se comenta en detalle en el capitulo siguiente. Si la frecuencia fundamental devuelta por la función es igual a cero no se producen cambios en las coordenadas del helicóptero. Además, se ha decidido que para frecuencias por debajo de 60 Hz el helicóptero no baje más de la parte inferior de la pantalla y que para frecuencias superiores a 280 Hz no supere la parte superior de la pantalla. Para las frecuencias intermedias se mueve el helicóptero a una posición proporcional con respecto a la altura de la pantalla. Sabiendo que se tiene 480 píxeles de alto y que la frecuencia mínima que se toma en cuenta es 60 Hz y la máxima 280 Hz, se realiza el siguiente cálculo para obtener la posición en píxeles Pos de las coordenadas verticales del helicóptero, donde f es la frecuencia detectada:

$$Pos = 480 - (f - 60) \times \frac{24}{11}$$

Ecuación 1: Fórmula de la ecuación que permite sacar la coordenada vertical del helicóptero

Explicación gráfica de la fórmula utilizada para el posicionamiento del helicóptero en la Figura 39:

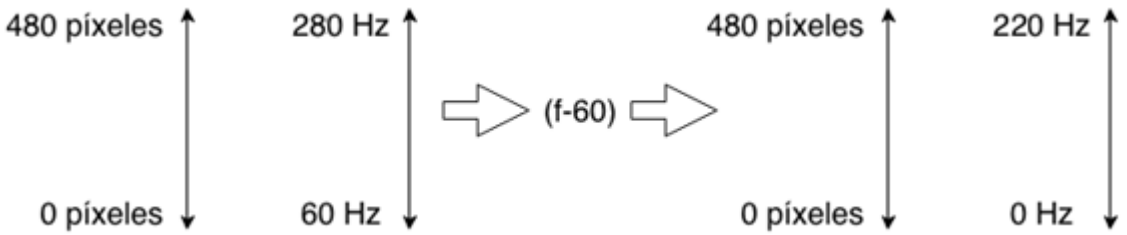

Figura 39: Explicación gráfica de la obtención de la ecuación

Se aplica una regla de tres y se obtiene la expresión:

$$(f - 60) \times \frac{480}{220}$$

Ecuación 2: Ecuación de la regla de tres que permite sacar la posición del helicóptero

El origen de coordenadas del motor de juego Andengine se encuentra en la parte superior izquierda de la pantalla por lo tanto debemos de restar lo anterior a la altura total de la pantalla en píxeles. Después de simplificar se halla la expresión final anterior.

Siguiendo con lo que se ejecuta en el *update handler*, el contador que se ha creado con anterioridad también marca la frecuencia fundamental que devuelve la función `frequency` cada vez que se actualiza el motor de juego. En este caso aunque la frecuencia sea inferior a los 60 Hz o rebase los 280 Hz, el contador muestra el valor que devuelve `frequency`. Se muestra un pseudocódigo a continuación que permite entender mejor lo sucedido en el *update handler*.

```
Inicio
   f=frequency
   si f es distinto de 0
      si f<60
         desplazar helicóptero a la parte inferior de
la pantalla
      si f>280
         desplazar helicóptero a la parte superior de
la pantalla
      si 60<f<280
         desplazar helicóptero a la posición
480-(f-60)*24/11
      escribir valor de f en pantalla
Fin
```

En la Figura 40 y Figura 41, se muestran dos pantallazos del juego ya en funcionamiento para ilustrar todo lo explicado. Uno de los pantallazos para cuando se detecta una frecuencia alta y otro para una frecuencia baja.

Figura 40: Pantallazo del juego de pitch al detectar una frecuencia de 241 Hz

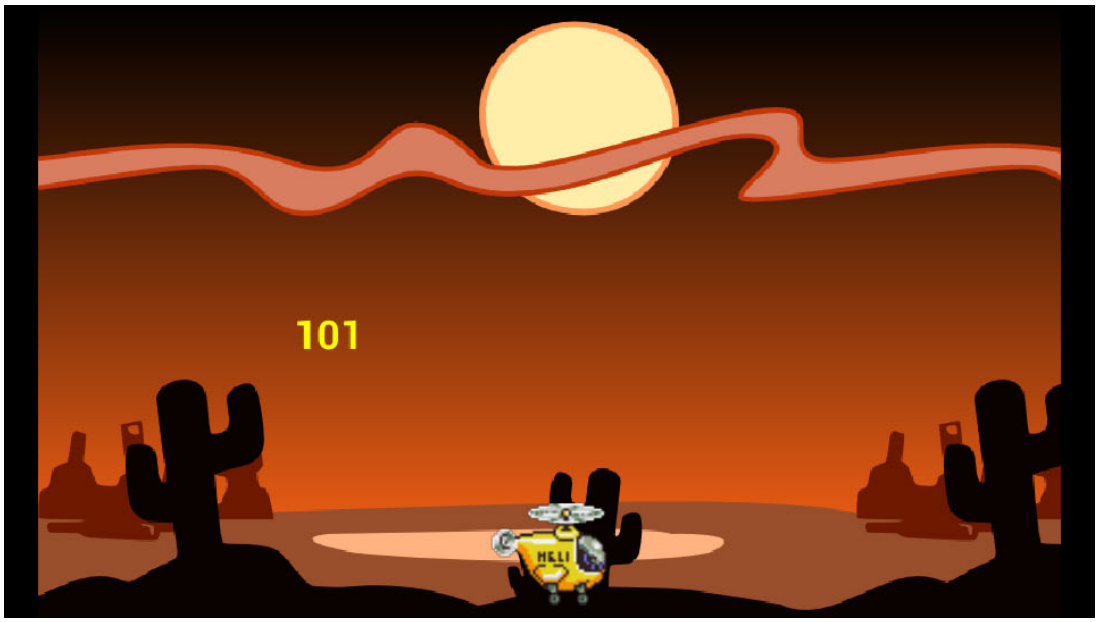

Figura 41: Pantallazo del juego de pitch al detectar una frecuencia de 101 Hz

- `createGame2Scene`

  Esta función se inicializa al pulsar el botón del juego de sonidos sonoros y sordos del menú. Tal y como se hizo en la función anterior se crea, en primer lugar, una nueva escena para este juego.

  Al contrario que en el juego anterior en este se define un color para el fondo de la pantalla. En el juego de pitch se utilizó una imagen como fondo pero en este juego se elige colorear la pantalla de un color verdoso.

  Una vez hecho esto se coloca el *sprite* que representa la cara que cambia de expresión dependiendo del sonido detectado. Se le posiciona en el centro de la pantalla, se multiplica por dos su tamaño para que sea más visible, y finalmente se le ancla a la escena.

  En esta función también se inicializa la grabación y usa un *update handler* para actualizar la escena en cada *frame*. En dicho *update handler* se hace una llamada a la función `sonorosordo` que identifica si el sonido es sonoro, sordo o inferior al umbral del nivel de intensidad elegido.

  En el caso en el que el sonido entrante por el micrófono tenga muy bajo nivel de intensidad aparece una cara sin rasgos faciales. Si el sonido es sordo la cara sonríe con la boca cerrada y si se detecta un sonido sonoro abre la boca. El pseudocódigo siguiente ilustra lo comentado.

```
Inicio
   s=sonorosordo
   si s está por debajo del umbral
      mostrar la cara sin expresión
   si s es sordo
      mostrar la cara sonriente
   si s es sonoro
      mostrar la cara con la boca
abierta
Fin
```

A continuación se visualizan en la Figura 42, Figura 43 y Figura 44, los tres pantallazos que corresponden a cada una de las tres posibles situaciones.

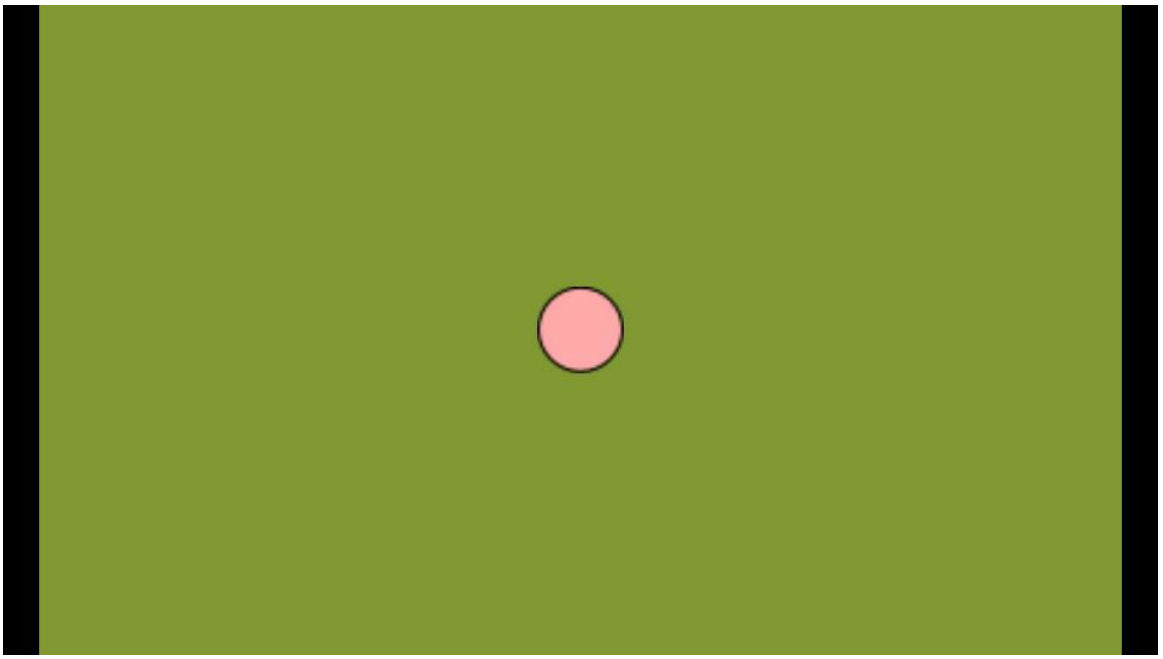

Figura 42: Pantallazo del juego de sonidos sonoros y sordos cuando el nivel del sonido detectado está por debajo del umbral

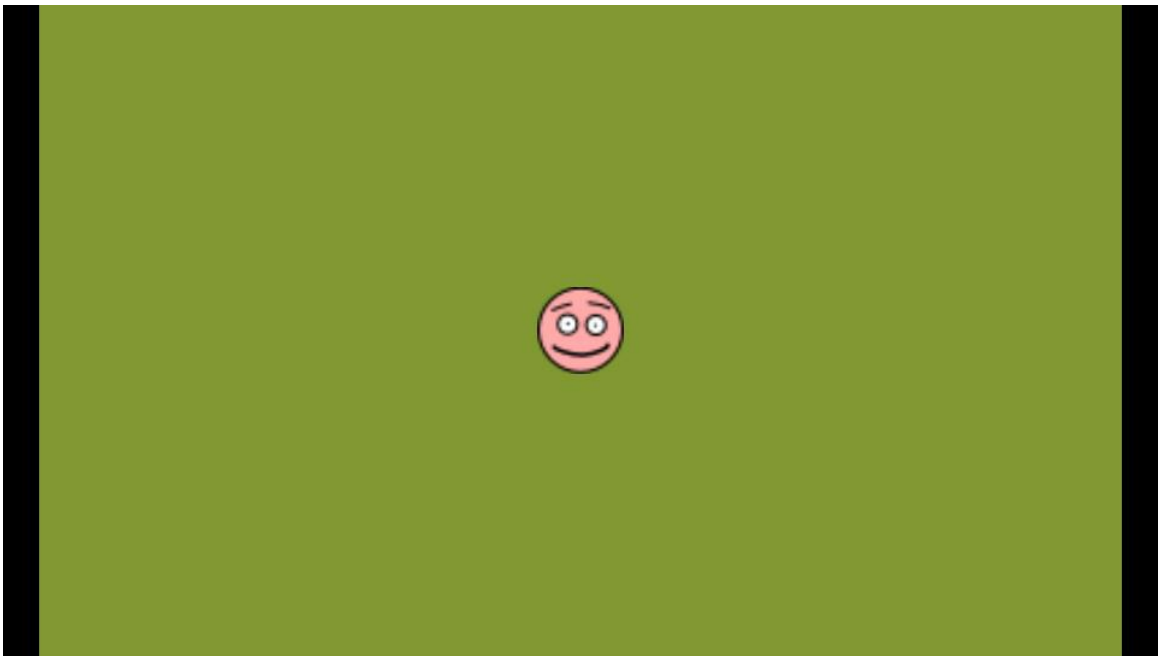

Figura 43: Pantallazo del juego de sonidos sonoros y sordos cuando el sonido detectado es sordo

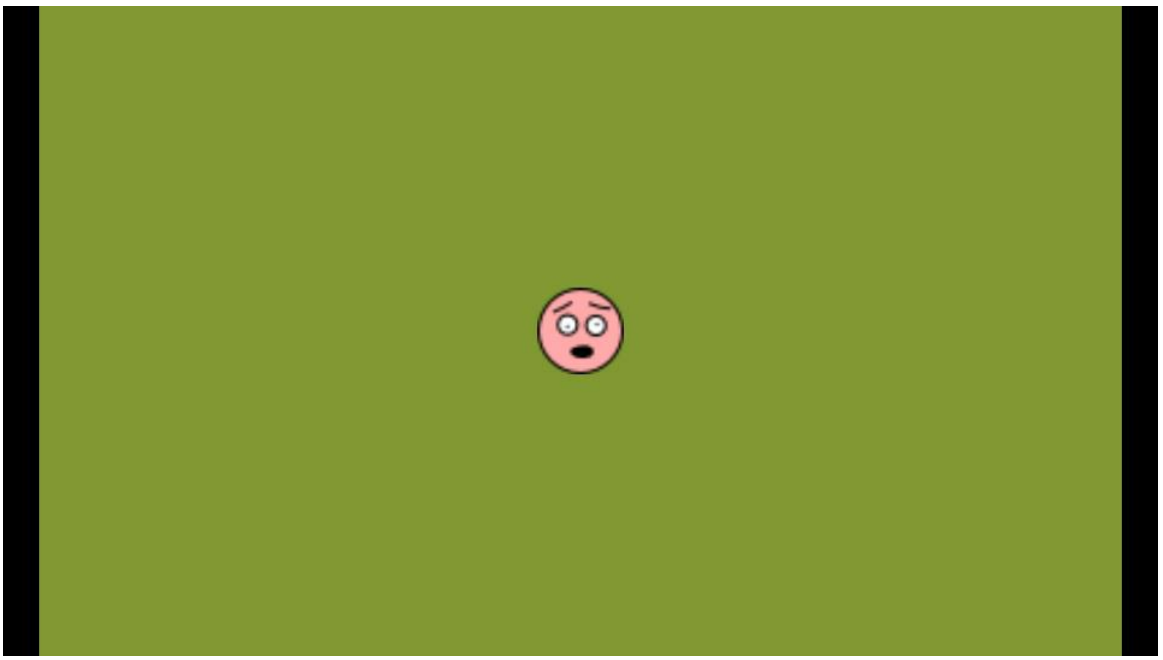

Figura 44: Pantallazo del juego de sonidos sonoros y sordos cuando el sonido detectado es sonoro

- `createGame3Scene`

Este juego tiene un funcionamiento muy similar al anterior por lo que la función `createGame3Scene` se asemeja mucho a `createGame2Scene`.

Se vuelve a crear una escena nueva para este juego. También se define un fondo colorado que en este caso es de un tono rojo. El *sprite* se coloca una vez más en el centro de la pantalla, se amplía por dos su tamaño y se ancla a la escena.

Después de inicializar la grabación se crea un update handler que llama a la función energía. Esta función determina si el sonido producido por el usuario supera el umbral determinado o no. Si el nivel es superado la cara cuadrada abre la boca y en caso contrario la mantiene cerrada con una sonrisa. Se puede apreciar en los pantallazos de la Figura 45 y Figura 46, y en el pseudocódigo siguiente.

```
Inicio
   s=energía
   si s es inferior al umbral
      mostrar la cara sonriente
   si s supera el umbral
      mostrar la cara con la boca
abierta
Fin
```

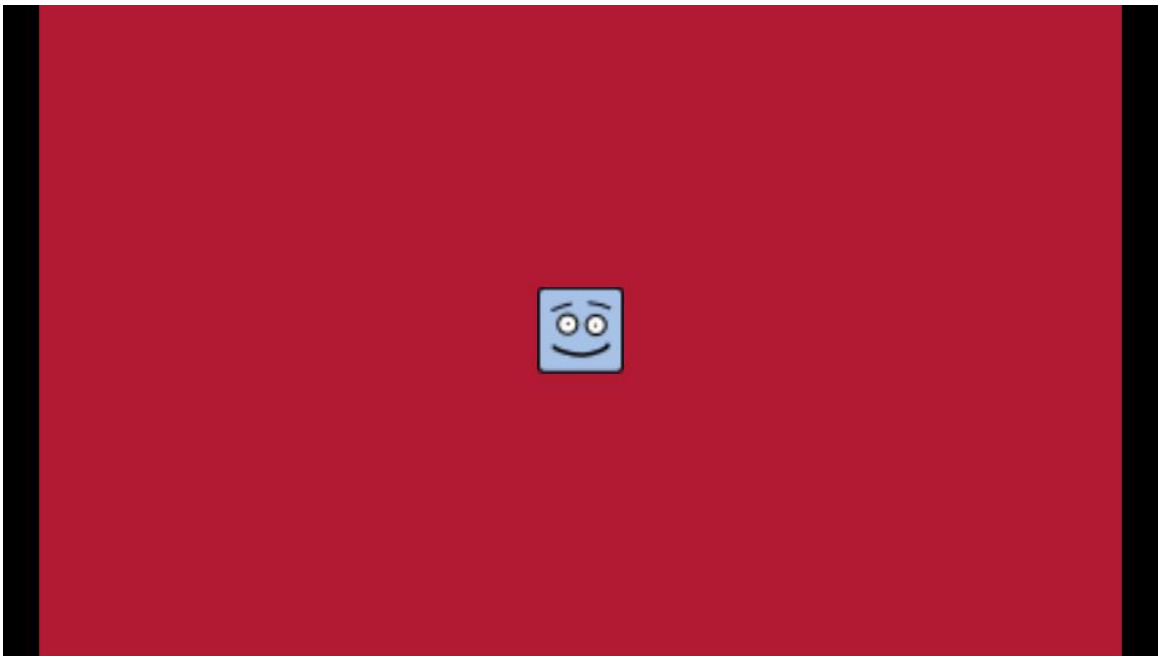

Figura 45: Pantallazo del juego de intensidad cuando el sonido es inferior al umbral

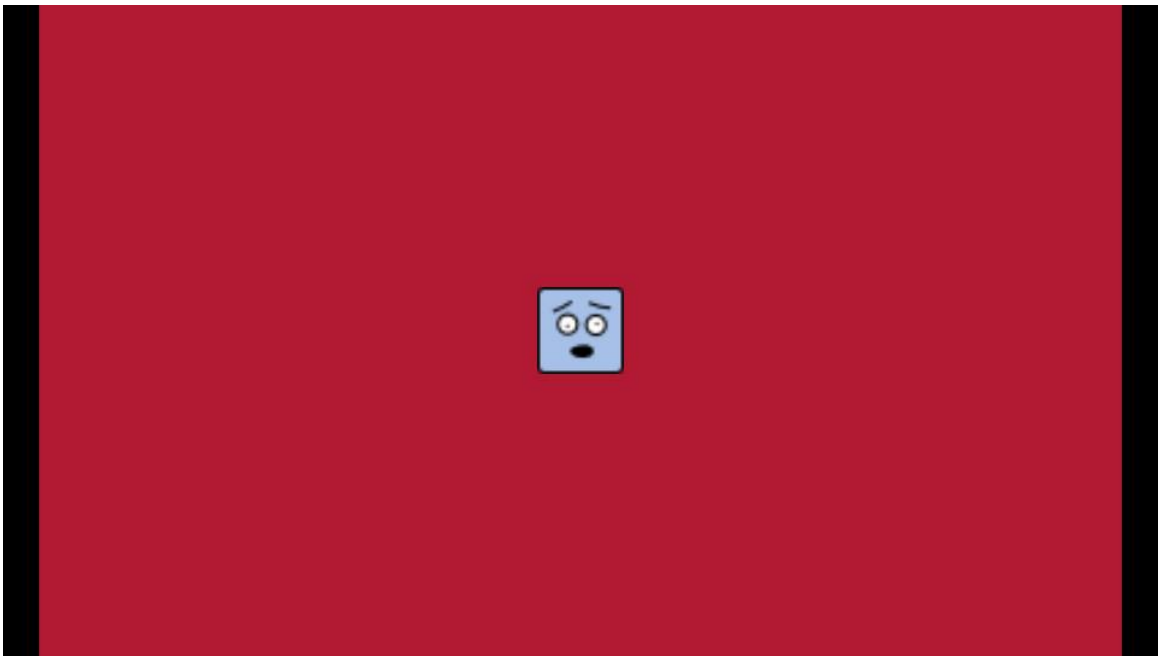

Figura 46: Pantallazo del juego de intensidad cuando el sonido es superior al umbral

- **`onBackPressed`**

  Este método es el último de la actividad SplashActivity. Es importante ya que permite, al pulsar el botón de "ir hacia atrás" del dispositivo móvil, desplazarse a la escena del menú de la aplicación (si se está en uno de los juegos) o bien salir del juego (si se está justamente en el menú).

  Para saber en qué escena se encuentra el usuario se utiliza un *getter* que en este caso es la función `getCurrentScene` que se encuentra en la actividad SceneManager. Esta función devuelve la escena en la que se está.

  Por lo tanto, en el caso en el que se esté en el menú y se pulse el botón de "ir hacia atrás", se decide que la aplicación finalice. En el caso en el que se encuentre el usuario en una de las tres escenas correspondientes a los juegos, se establece la escena del menú gracias a la función vista con anterioridad `setCurrentScene`.

Para finalizar este apartado se presenta a continuación en la Figura 47 el diagrama UML del software completo desarrollado para esta aplicación.

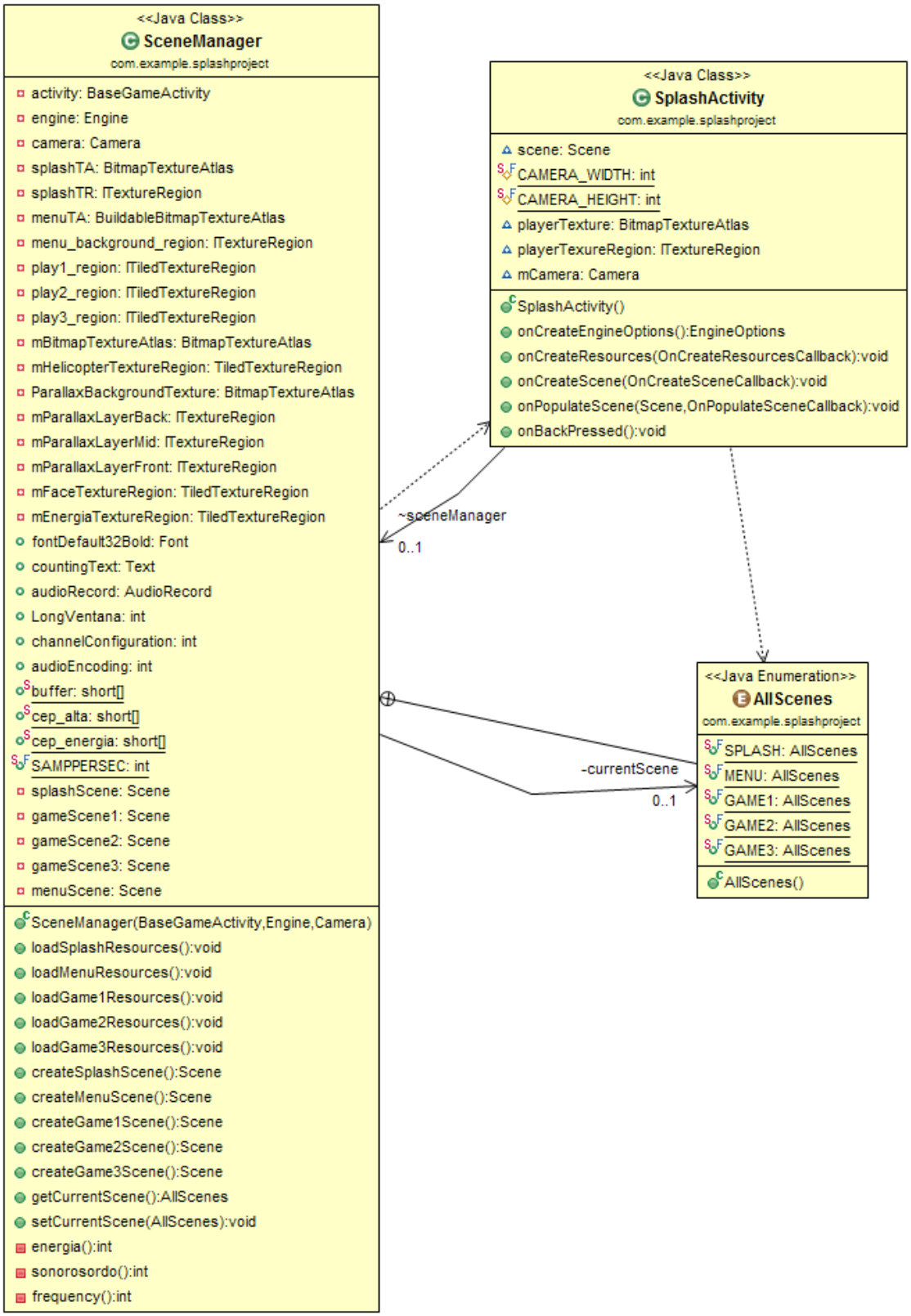

Figura 47: Diagrama UML del software desarrollado

4.2.3. Sistema de captura y análisis de audio

En este apartado se explica la captura del audio procedente del micrófono incorporado en el dispositivo móvil así como el funcionamiento de las tres funciones de procesado de audio específicas de cada uno de los tres juegos. La función implementada para la detección de la frecuencia fundamental de la voz del usuario ha sido llamada `frequency`, para la detección de sonidos sonoros y sordos se ha creado una función llamada `sonorosordo` y para el juego de intensidad se tiene la función `energía`. Cada una de ellas se basa en unos algoritmos de audio que se comentan a continuación.

- **`frequency`**

  Para hallar la frecuencia fundamental de la voz del usuario se ha elegido el método cepstrum. Este método consiste en calcular la transformada de Fourier del espectro de la señal capturada en escala logarítmica.

  Primero se calcula la FFT, luego se obtiene el logaritmo del resultado y finalmente se calcula la transformada inversa de Fourier. Al detectar un gran pico en el cepstrum calculado se permite hallar el *pitch* de dicho segmento. Esto se realiza dividiendo la frecuencia de muestreo por la posición de dicho máximo en el segmento.

  Tal y como se hará en las otras dos funciones de procesado de audio, lo primero que se hace en `frequency` es definir un bucle infinito en el que se leen las muestras de audio capturadas por el micrófono y se guardan en el *buffer* que se ha creado para ello. A continuación se busca el periodo de *pitch* del segmento leído sin que se salga del rango de 2 ms y 15 ms. Seguidamente se crea una ventana de tipo Hamming y se multiplica el segmento leído con ella. Haciendo uso de la librería JTransforms se realiza una transformada de Fourier de la ventana resultante anterior. A este nuevo tramo al que se le ha aplicado la FFT se le aplica la función logaritmo para a continuación hacer una transformada de Fourier inversa. Se buscan máximos que tengan amplitud suficiente para finalmente quedarse con el último que se encuentra que supere el umbral que se ha elegido. Para hallar la frecuencia fundamental se divide la frecuencia de muestreo por el número de la muestra en el que se ha encontrado el máximo anterior y se devuelve el valor encontrado. A continuación se presenta el código de Matlab del que proviene el algoritmo creado.

```
% Fs es la frecuencia de muestreo
% f es el pitch obtenido a partir del cepstrum
% se busca el periodo de pitch entre los valores de 2ms
% y 15ms

Tmin=round(0.002*Fs);
Tmax=round(0.015*Fs);

UMBRAL_ENER=20;
tam_vent=length(segmento);

% Para hacer fft's rapidas cogemos la potencia de dos superior
tam_fft=2^( ceil( log2(tam_vent)));

% ventana es la voz enventanada usando Hamming
ventana=segmento.* hamming(tam_vent)';

cepstrum = real( ifft( log( abs( fft(ventana, tam_fft))), tam_fft) );
cepstrum = cepstrum(1:tam_fft/2);
cep_alta = cepstrum(Tmin:tam_fft/2);
cep_energia = cep_alta.^2;

[maximo,i_max]=max(cep_energia(1:Tmax-Tmin));

if (maximo>UMBRAL_ENER*mean(cep_energia)),

   % Se buscan máximos anteriores (de índice menor) que tengan amplitud
% suficiente,
   % quedándonos con el último que encontremos (el primer máximo que
% supera el umbral).
   [max2,i_max2]=max(cep_energia(1:i_max-Tmin));
   while max2>UMBRAL_ENER*mean(cep_energia)
      i_max=i_max2;
      [max2,i_max2]=max(cep_energia(1:i_max2-Tmin));
   end;

   % i_max+Tmin-1 es el numero de muestra dentro del vector cepstrum
   % Pero en Matlab la primera muestra (n=0) tiene índice 1, por lo que
% hay que restar 1
   T=i_max+Tmin-1-1;
   if T~=0,
      f=Fs/T;
   else
      f=0;
   end

else
   f=0;
end
```

- **sonorosordo**

La función `sonorosordo` permite diferenciar tres situaciones distintas. Una de ellas es si el nivel del audio es inferior a un cierto umbral. Las otras dos son en el caso en el que el nivel del audio supera dicho umbral, se detecta si el sonido es sonoro o sordo. La función devuelve un 0 si el sonido es inferior al umbral, un 1 si es sonoro y un 2 si es sordo.

La función empieza con un bucle en el que lo primero que se hace es leer los datos de audio. Se calcula la energía logarítmica del segmento para a continuación calcular el número de cruces por cero que existen en el *buffer* de audio leído. Si el nivel de audio es superior al umbral y la tasa de cruces por cero es superior a lo establecido se considera que el sonido es sordo. En cambio si el nivel del audio supera el umbral pero la tasa de cruces por cero calculada es inferior a lo establecido se considera que el sonido es sonoro.

```
% Umbrales para la decisión voz/silencio
UMBRAL_VOZ_ENER=63.5;
UMBRAL_VOZ_TCC=0.25;
S=0;
% mSegnal: es un vector fila que contiene las muestras a analizar.
% E: es la energia en escala logaritmica
% e: es la energia en escala lineal.

[frames, long]=size( mSegnal );
E=zeros( 1, frames );
e=zeros( 1, frames );

for i=1:frames,
     e(i)=(1/long)*sum( mSegnal(i,:).^2 );
     eps=1e-12;
     E(i)=10*log10( eps+e(i) );
end

long=length(mSegnal);
if long==0,
   TCC=0;
else
   TCC=0.5*sum(abs(sign(mSegnal(2:long))-sign(mSegnal(1:long-1))));
   TCC=TCC/long;
end

if E>UMBRAL_VOZ_ENER & TCC<UMBRAL_VOZ_TCC,
   S = 1;
end
if E>UMBRAL_VOZ_ENER & TCC>UMBRAL_VOZ_TCC,
   S = 2;
end
```

- **energia**

Esta función consiste en calcular la energía logarítmica de las muestras de audio capturadas por el micrófono y devolver un valor dependiendo de si el nivel del audio ha superado el umbral que se ha elegido. Si supera dicho umbral la función un 1 y en caso contrario devuelve un 0.

Ya en la función `energia`, se crea un bucle que se repetirá hasta que se salga del juego. En él lo primero que se hace es leer las muestras de audio recogidas por el micrófono gracias al objeto AudioRecord. A continuación se recorre el buffer con las muestras de audio para realizar los cálculos pertinentes que permiten hallar la energía del segmento de datos leído. Una vez que se obtiene la energía calculada se compara con el umbral elegido.

```
% Umbral de energia
UMBRAL_VOZ_ENER=63.5;
s=0;
% mSegnal: es un vector fila que contiene las muestras a analizar.
% E: es la energia en escala logaritmica
% e: es la energia en escala lineal.

[frames, long]=size( mSegnal );
E=zeros( 1, frames );
e=zeros( 1, frames );


for i=1:frames,
     e(i)=(1/long)*sum( mSegnal(i,:).^2 );
     eps=1e-12;
     E(i)=10*log10( eps+e(i) );
end

E(i)= E(i)/frames;

if E>UMBRAL_VOZ_ENER,
       s=1;
end
```

Capítulo 5: Conclusiones y líneas de trabajo futuras

Se ha desarrollado una aplicación que permite al usuario entrenar ciertos parámetros de su voz como son el *pitch*, la intensidad y los sonidos sordos y sonoros. Todo ello, ofreciendo al usuario la oportunidad de realizar su rehabilitación vocal con comodidad desde su dispositivo móvil. Además de entrenarse de forma amena mediante los juegos que se han diseñado, estos ejercicios ofrecen un *feedback* visual en tiempo real.

Como mejoras posibles para la aplicación, se podrían haber añadido elementos en los juegos, como puntuaciones y objetivos concretos, para dinamizar la experiencia y entrenamiento vocal del paciente.

En lo que a aplicaciones para la rehabilitación de la voz controladas por voz en dispositivos Android se refiere, vale la pena destacar que no se han desarrollado aplicaciones de este tipo para esta plataforma. Esto conlleva a que sea un terreno interesante en el que seguir trabajando, ya que la oferta de estas aplicaciones es inexistente en el momento en el que se ha realizado este trabajo.

Apéndice A

Instalación de Eclipse IDE y Android SDK

Para ello visitamos la página web: http://developer.android.com/sdk/index.html
Aquí podremos descargar el ADT Bundle, que incluye todos los archivos necesarios para poder empezar a desarrollar aplicaciones Android. Este paquete contiene:

- Eclipse con el plugin ADT
- Las herramientas de Android SDK
- Las herramientas de la plataforma Android
- La versión más reciente de la plataforma Android
- La versión más reciente del sistema de imágenes Android para el emulador

Después de descargar el ADT Bundle se descomprime y guarda en un directorio.
A partir de este momento Eclipse está operativo y listo para desarrollar aplicaciones Android.

Instalación del motor de juego Andengine

Una vez que Eclipse IDE y Android SDK estén debidamente instalados es necesario incorporar Andengine a nuestro proyecto. Andengine se aloja en GitHub, un servicio web que reúne proyectos de forma pública. El motor de juego en cuestión se encuentra en el dominio siguiente: https://github.com/nicolasgramlich/AndEngine

Andengine usa la versión Android SDK 4.0 por lo que hay que asegurarse que esta plataforma está debidamente instalada. Para ello se selecciona en la barra de herramientas de Eclipse la opción “Window” y a continuación “Android SDK Manager”. Las herramientas SDK deben estar actualizadas por encima del nivel 17 para que el motor de juego funcione. Además se selecciona la casilla correspondiente a la versión de Android más reciente (Android 4.4 API 19 durante la elaboración de este proyecto).

A continuación, se selecciona con el botón derecho del ratón “Package Explorer” dentro de la sección de trabajo y se elige la opción “Import”. Se selecciona “Projects from Git” dentro de la carpeta Git, elegimos URI y pulsamos “Next”. Se copia en la página siguiente la dirección https://github.com/nicolasgramlich/AndEngine.git en el espacio que le corresponde. Después de darle a “Next” dos veces, hay que asegurarse de que GLES2 se ha seleccionado en la sección “Initial branch”. Se pulsa “Next” hasta que la instalación de Andengine en nuestro proyecto se haya realizado.

Una vez añadida la librería Andengine en el *workspace* se le añade el proyecto del juego que se quiere crear. Se selecciona "Package Explorer" con el botón derecho del ratón y se clica en "New" seguido de "Android Project". En este paso se puede nombrar el proyecto. Se elige Android 2.2 como plataforma diana para luego cliquear en "Next" y darle nombre al "Package Name". En el siguiente paso ya se puede crear la actividad del juego.

Finalmente hay que permitir al juego que se está creando poder usar la librería Andengine que se ha instalado anteriormente. Para ello se selecciona en el *workspace* con el botón derecho la carpeta contenedora del juego recientemente creado para luego darle a "Properties" y "Android". Se clica en "Add…" dentro de la sección "Library" y se selecciona la librería Andengine que se ha instalado. Validar los cambios realizados. Para acabar con la todavía abierta ventana de propiedades ir a "Java Build Path" y luego a "Order and Export" y seleccionar todas las casillas.

Instalación de la librería JTransforms

Lo primero que hay que hacer es visitar la web en la que se encuentra esta librería de FFT y descargarla en: https://sites.google.com/site/piotrwendykier/software/jtransforms

A continuación hay que instalar esta librería externa con extensión ".jar". Para ello hay que ubicarse en el proyecto al que se quiere añadir la librería y clicar con el botón derecho sobre él y seleccionar "Build Path" seguido de "Configure Build Path". A continuación aparece una ventana emergente en la que hay que seleccionar "Libraries". Ahora sólo queda clicar en el botón "Add External Jars…" y seleccionar la librería JTransforms.

Apéndice B

com.example.splashproject

Class SceneManager

java.lang.Object
    com.example.splashproject.SceneManager

```
public class SceneManager
extends java.lang.Object
```

Esta clase controla la navegación por la aplicación y sirve para cargar los recursos de los juegos

Version:

1.0, Agosto 2014

Author:

John Ireland

Nested Class Summary

Nested Classes

Modifier and Type	Class and Description
static class	SceneManager.AllScenes Sirve para manejar las escenas de la aplicación

Field Summary

Fields

Modifier and Type	Field and Description
int	audioEncoding
AudioRecord	audioRecord
static short[]	buffer
static short[]	cep_alta
static short[]	cep_energia
int	channelConfiguration
org.andengine.entity.text.Text	countingText
org.andengine.opengl.font.Font	fontDefault32Bold
int	LongVentana
static int	SAMPPERSEC

Constructor Summary

Constructors

Constructor and Description
SceneManager(org.andengine.ui.activity.BaseGameActivity act, org.andengine.engine.Engine eng, org.andengine.engine.camera.Camera cam) Controla la navegación del juego

Method Summary

Methods

Modifier and Type	Method and Description
org.andengine.entity.scene.Scene	createGame1Scene() Crea la escena del juego de pitch
org.andengine.entity.scene.Scene	createGame2Scene() Crea la escena del juego de sonidos sonoros y sordos
org.andengine.entity.scene.Scene	createGame3Scene() Crea la escena del juego de intensidad
org.andengine.entity.scene.Scene	createMenuScene() Crea la escena de la pantalla del menú
org.andengine.entity.scene.Scene	createSplashScene() Crea la escena de la pantalla de arranque
SceneManager.AllScenes	getCurrentScene()
void	loadGame1Resources() Carga los recursos del juego de pitch
void	loadGame2Resources() Carga los recursos del juego de sonidos sonoros y sordos
void	loadGame3Resources() Carga los recursos del juego de intensidad
void	loadMenuResources() Carga los recursos del menú
void	loadSplashResources() Carga los recursos de la pantalla de arranque
void	setCurrentScene(SceneManager.AllScenes currentScene) Establece una de las escenas de la aplicación

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

fontDefault32Bold

```
public org.andengine.opengl.font.Font fontDefault32Bold
```

countingText

```
public org.andengine.entity.text.Text countingText
```

audioRecord

```
public AudioRecord audioRecord
```

LongVentana

```
public int LongVentana
```

channelConfiguration

```
public int channelConfiguration
```

audioEncoding

```
public int audioEncoding
```

buffer

```
public static short[] buffer
```

cep_alta

```
public static short[] cep_alta
```

cep_energia

```
public static short[] cep_energia
```

SAMPPERSEC

```
public static final int SAMPPERSEC
```

See Also:

Constant Field Values

Constructor Detail

SceneManager

```
public SceneManager(org.andengine.ui.activity.BaseGameActivity act,
            org.andengine.engine.Engine eng,
            org.andengine.engine.camera.Camera cam)
```

Controla la navegación del juego

Parameters:

`act` - actividad principal

`eng` - motor de juego

`cam` - cámara del motor de juego

Method Detail

loadSplashResources

```
public void loadSplashResources()
```

Carga los recursos de la pantalla de arranque

loadMenuResources

```
public void loadMenuResources()
```

Carga los recursos del menú

loadGame1Resources

```
public void loadGame1Resources()
```

Carga los recursos del juego de pitch

loadGame2Resources

```
public void loadGame2Resources()
```

Carga los recursos del juego de sonidos sonoros y sordos

loadGame3Resources

```
public void loadGame3Resources()
```

Carga los recursos del juego de intensidad

createSplashScene

```
public org.andengine.entity.scene.Scene createSplashScene()
```

Crea la escena de la pantalla de arranque

Returns:

devuelve la escena de la pantalla de arranque

createMenuScene

```
public org.andengine.entity.scene.Scene createMenuScene()
```

Crea la escena de la pantalla del menú

Returns:

devuelve la escena de la pantalla del menú

createGame1Scene

```
public org.andengine.entity.scene.Scene createGame1Scene()
```

Crea la escena del juego de pitch

Returns:

devuelve la escna del juego de pitch

createGame2Scene

```
public org.andengine.entity.scene.Scene createGame2Scene()
```

Crea la escena del juego de sonidos sonoros y sordos

Returns:

devuelve la escena del juego de sonidos sonoros y sordos

createGame3Scene

```
public org.andengine.entity.scene.Scene createGame3Scene()
```

Crea la escena del juego de intensidad

Returns:

devuelve la escena del juego de intensidad

getCurrentScene

```
public SceneManager.AllScenes getCurrentScene()
```

Returns:

devuelve la escena actual

setCurrentScene

```
public void setCurrentScene(SceneManager.AllScenes currentScene)
```

Establece una de las escenas de la aplicación

Parameters:

`currentScene` - es la escena actual

Código

SplashActivity.java

```
package com.example.splashproject;

import org.andengine.engine.camera.Camera;
import org.andengine.engine.handler.timer.ITimerCallback;
import org.andengine.engine.handler.timer.TimerHandler;
import org.andengine.engine.options.EngineOptions;
import org.andengine.engine.options.ScreenOrientation;
import
org.andengine.engine.options.resolutionpolicy.RatioResolutionPolicy;
import org.andengine.entity.scene.Scene;
import org.andengine.opengl.texture.atlas.bitmap.BitmapTextureAtlas;
import org.andengine.opengl.texture.region.ITextureRegion;
import org.andengine.ui.activity.BaseGameActivity;

import com.example.splashproject.SceneManager.AllScenes;

/**
 * @author John Ireland
 * @version 1.0, Agosto 2014
 */
public class SplashActivity extends BaseGameActivity {

    Scene scene;
    protected static final int CAMERA_WIDTH = 800;
    protected static final int CAMERA_HEIGHT = 480;
    BitmapTextureAtlas playerTexture;
    ITextureRegion playerTexureRegion;

    SceneManager sceneManager;
    Camera mCamera;

    @Override
    public EngineOptions onCreateEngineOptions() {

        mCamera = new Camera(0, 0, CAMERA_WIDTH, CAMERA_HEIGHT);

        EngineOptions options = new EngineOptions(true,
                ScreenOrientation.LANDSCAPE_FIXED, new
RatioResolutionPolicy(
                        CAMERA_WIDTH, CAMERA_HEIGHT), mCamera);
        return options;
    }

    @Override
    public void onCreateResources(
            OnCreateResourcesCallback pOnCreateResourcesCallback)
            throws Exception {
        sceneManager = new SceneManager(this, mEngine, mCamera);
        sceneManager.loadSplashResources();
```

```
        pOnCreateResourcesCallback.onCreateResourcesFinished();
    }

    @Override
    public void onCreateScene(OnCreateSceneCallback
pOnCreateSceneCallback)
            throws Exception {
        pOnCreateSceneCallback.onCreateSceneFinished(sceneManager
                .createSplashScene());
    }

    @Override
    public void onPopulateScene(Scene pScene,
            OnPopulateSceneCallback pOnPopulateSceneCallback) throws
Exception {

        mEngine.registerUpdateHandler(new TimerHandler(3f,
                new ITimerCallback() {

                    @Override
                    public void onTimePassed(TimerHandler pTimerHandler) {
                        mEngine.unregisterUpdateHandler(pTimerHandler);
                        // TODO Auto-generated method stub
                        sceneManager.loadMenuResources();
                        sceneManager.loadGame1Resources();
                        sceneManager.loadGame2Resources();
                        sceneManager.loadGame3Resources();
                        sceneManager.createMenuScene();
                        sceneManager.setCurrentScene(AllScenes.MENU);
                    }
                }));



        pOnPopulateSceneCallback.onPopulateSceneFinished();
    }

    @Override
    public void onBackPressed() {
        if (sceneManager.getCurrentScene() == AllScenes.MENU)
        {
        finish();}
        if (sceneManager.getCurrentScene() == AllScenes.GAME1 ||
sceneManager.getCurrentScene() == AllScenes.GAME2 ||
sceneManager.getCurrentScene() == AllScenes.GAME3)
        {
        sceneManager.setCurrentScene(AllScenes.MENU);}

    }

}
```

SceneManager.java

```
package com.example.splashproject;

import org.andengine.engine.Engine;
import org.andengine.engine.camera.Camera;
import org.andengine.engine.handler.IUpdateHandler;
import org.andengine.entity.modifier.MoveYModifier;
import org.andengine.entity.scene.Scene;
import org.andengine.entity.scene.background.AutoParallaxBackground;
import org.andengine.entity.scene.background.Background;
import
org.andengine.entity.scene.background.ParallaxBackground.ParallaxEntity;
import org.andengine.entity.scene.background.SpriteBackground;
import org.andengine.entity.sprite.AnimatedSprite;
import org.andengine.entity.sprite.Sprite;
import org.andengine.entity.sprite.TiledSprite;
import org.andengine.entity.text.Text;
import org.andengine.input.touch.TouchEvent;
import org.andengine.opengl.font.Font;
import org.andengine.opengl.font.FontFactory;
import org.andengine.opengl.texture.TextureOptions;
import org.andengine.opengl.texture.atlas.bitmap.BitmapTextureAtlas;
import
org.andengine.opengl.texture.atlas.bitmap.BitmapTextureAtlasTextureRegio
nFactory;
import
org.andengine.opengl.texture.atlas.bitmap.BuildableBitmapTextureAtlas;
import
org.andengine.opengl.texture.atlas.bitmap.source.IBitmapTextureAtlasSour
ce;
import
org.andengine.opengl.texture.atlas.buildable.builder.BlackPawnTextureAtl
asBuilder;
import
org.andengine.opengl.texture.atlas.buildable.builder.ITextureAtlasBuilde
r.TextureAtlasBuilderException;
import org.andengine.opengl.texture.region.ITextureRegion;
import org.andengine.opengl.texture.region.ITiledTextureRegion;
import org.andengine.opengl.texture.region.TiledTextureRegion;
import org.andengine.opengl.util.GLState;
import org.andengine.ui.activity.BaseGameActivity;
import org.andengine.util.color.Color;
import org.andengine.util.debug.Debug;

import android.graphics.Typeface;
import android.media.AudioFormat;
import android.media.AudioRecord;

import edu.emory.mathcs.jtransforms.fft.DoubleFFT_1D;
/**
 * Esta clase controla la navegación por la aplicación y sirve para cargar
los recursos de los juegos
 * @author John Ireland
 * @version 1.0, Agosto 2014
```

```
 */
public class SceneManager {
    private AllScenes currentScene;
    private BaseGameActivity activity;
    private Engine engine;
    private Camera camera;
    //splash
    private BitmapTextureAtlas splashTA;
    private ITextureRegion splashTR;
    //menu
    private BuildableBitmapTextureAtlas menuTA;
    private ITextureRegion menu_background_region;
    private ITiledTextureRegion play1_region;
    private ITiledTextureRegion play2_region;
    private ITiledTextureRegion play3_region;

    //Juego1
        private BitmapTextureAtlas mBitmapTextureAtlas;
        private TiledTextureRegion mHelicopterTextureRegion;

        private BitmapTextureAtlas ParallaxBackgroundTexture;
        private ITextureRegion mParallaxLayerBack;
        private ITextureRegion mParallaxLayerMid;
        private ITextureRegion mParallaxLayerFront;
    //Juego2
        private TiledTextureRegion mFaceTextureRegion;
    //Juego3
        private TiledTextureRegion mEnergiaTextureRegion;

    //texto
        public Font fontDefault32Bold;
        public Text countingText;

    //parametros de audio
        public AudioRecord audioRecord;
        public int LongVentana = 2048;
        public int channelConfiguration = AudioFormat.CHANNEL_IN_MONO;
        public int audioEncoding = AudioFormat.ENCODING_PCM_16BIT;
        public static short[] buffer; //+-32767
        public static short[] cep_alta;
        public static short[] cep_energia;
        public static final int SAMPPERSEC = 22500;

    private Scene splashScene, gameScene1, gameScene2, gameScene3,
menuScene;
/**
 * Sirve para manejar las escenas de la aplicación
 *
 */
    public enum AllScenes {
        SPLASH, MENU, GAME1, GAME2, GAME3
    }
/**
 * Controla la navegación del juego
 * @param act actividad principal
 * @param eng motor de juego
 * @param cam cámara del motor de juego
```

```
 */
    public SceneManager(BaseGameActivity act, Engine eng, Camera cam) {
        // TODO Auto-generated constructor stub
        this.activity = act;
        this.engine = eng;
        this.camera = cam;
    }
/**
 * Carga los recursos de la pantalla de arranque
 */
    public void loadSplashResources() {
        BitmapTextureAtlasTextureRegionFactory.setAssetBasePath("gfx/");
        splashTA = new
BitmapTextureAtlas(this.activity.getTextureManager(),
                512, 256, TextureOptions.BILINEAR);
        splashTR = BitmapTextureAtlasTextureRegionFactory.createFromAsset(
                splashTA, this.activity, "entrena_splash.png", 0, 0);
        splashTA.load();
    }
/**
 * Carga los recursos del menú
 */
    public void loadMenuResources() {
        BitmapTextureAtlasTextureRegionFactory.setAssetBasePath("gfx/");
        menuTA = new
BuildableBitmapTextureAtlas(this.activity.getTextureManager(),
                1024, 1024, TextureOptions.BILINEAR);
        menu_background_region =
BitmapTextureAtlasTextureRegionFactory.createFromAsset(menuTA, activity,
"menu_background.png");
        play1_region =
BitmapTextureAtlasTextureRegionFactory.createTiledFromAsset(menuTA,
activity, "pitch_tiled.png", 2, 1);
        play2_region =
BitmapTextureAtlasTextureRegionFactory.createTiledFromAsset(menuTA,
activity, "sonorosordo_tiled.png", 2, 1);
        play3_region =
BitmapTextureAtlasTextureRegionFactory.createTiledFromAsset(menuTA,
activity, "intensidad_tiled.png", 2, 1);
        try
    {
            this.menuTA.build(new
BlackPawnTextureAtlasBuilder<IBitmapTextureAtlasSource,
BitmapTextureAtlas>(0, 1, 0));
            this.menuTA.load();
        }
    catch (final TextureAtlasBuilderException e)
    {
            Debug.e(e);
        }

    }
/**
 * Carga los recursos del juego de pitch
 */
    public void loadGame1Resources() {
        //font
```

```
        fontDefault32Bold = FontFactory.create(
        engine.getFontManager(),
        engine.getTextureManager(), 256, 256,
        Typeface.create(Typeface.DEFAULT, Typeface.BOLD),
        32f, true, Color.YELLOW_ARGB_PACKED_INT);
        fontDefault32Bold.load();

//sprite
        BitmapTextureAtlasTextureRegionFactory.setAssetBasePath("gfx/");

        this.mBitmapTextureAtlas = new
BitmapTextureAtlas(this.activity.getTextureManager(), 94, 40,
TextureOptions.BILINEAR);
        this.mHelicopterTextureRegion =
BitmapTextureAtlasTextureRegionFactory.createTiledFromAsset(this.mBitmap
TextureAtlas, this.activity, "helicopter.png", 0, 0, 2, 1);
        this.mBitmapTextureAtlas.load();

        this.ParallaxBackgroundTexture = new
BitmapTextureAtlas(this.activity.getTextureManager(), 1024, 1024);
        this.mParallaxLayerFront =
BitmapTextureAtlasTextureRegionFactory.createFromAsset(this.ParallaxBack
groundTexture, this.activity, "parallax_background_layer_front.png", 0,
0);
        this.mParallaxLayerBack =
BitmapTextureAtlasTextureRegionFactory.createFromAsset(this.ParallaxBack
groundTexture, this.activity, "parallax_background_layer_back.png", 0,
188);
        this.mParallaxLayerMid =
BitmapTextureAtlasTextureRegionFactory.createFromAsset(this.ParallaxBack
groundTexture, this.activity, "parallax_background_layer_mid.png", 0,
669);
        this.ParallaxBackgroundTexture.load();

//audio
        buffer = new short[LongVentana];
       audioRecord = new
AudioRecord(android.media.MediaRecorder.AudioSource.MIC,SAMPPERSEC,chann
elConfiguration,audioEncoding,LongVentana);
///////

    }
/**
 * Carga los recursos del juego de sonidos sonoros y sordos
 */
   public void loadGame2Resources() {

        BitmapTextureAtlasTextureRegionFactory.setAssetBasePath("gfx/");

        this.mBitmapTextureAtlas = new
BitmapTextureAtlas(this.activity.getTextureManager(), 96, 32,
TextureOptions.BILINEAR);
        this.mFaceTextureRegion =
BitmapTextureAtlasTextureRegionFactory.createTiledFromAsset(this.mBitmap
TextureAtlas, this.activity, "cara_sonoro_sordo.png", 0, 0, 3, 1);
        this.mBitmapTextureAtlas.load();
```

```
        //audio
                buffer = new short[LongVentana];
                audioRecord = new
AudioRecord(android.media.MediaRecorder.AudioSource.MIC,SAMPPERSEC,chann
elConfiguration,audioEncoding,LongVentana);
        ///////
    }
/**
 *  Carga los recursos del juego de intensidad
 */
    public void loadGame3Resources() {

        BitmapTextureAtlasTextureRegionFactory.setAssetBasePath("gfx/");

        this.mBitmapTextureAtlas = new
BitmapTextureAtlas(this.activity.getTextureManager(), 64, 32,
TextureOptions.BILINEAR);
        this.mEnergiaTextureRegion =
BitmapTextureAtlasTextureRegionFactory.createTiledFromAsset(this.mBitmap
TextureAtlas, this.activity, "face_box_tiled.png", 0, 0, 2, 1);
        this.mBitmapTextureAtlas.load();

        //audio
                buffer = new short[LongVentana];
                audioRecord = new
AudioRecord(android.media.MediaRecorder.AudioSource.MIC,SAMPPERSEC,chann
elConfiguration,audioEncoding,LongVentana);
        ///////
    }

/**
 * Crea la escena de la pantalla de arranque
 * @return devuelve la escena de la pantalla de arranque
 */
    public Scene createSplashScene() {
        splashScene = new Scene();
        splashScene.setBackground(new Background(0, 0, 0));

        Sprite icon = new Sprite(0, 0, splashTR,
                engine.getVertexBufferObjectManager()){
                    @Override
                   protected void preDraw(GLState pGLState, Camera pCamera)
                {
                        super.preDraw(pGLState, pCamera);
                        pGLState.enableDither();
                    }
                };
        icon.setPosition((camera.getWidth() - icon.getWidth()) / 2,
                (camera.getHeight() - icon.getHeight()) / 2);
        splashScene.attachChild(icon);
        return splashScene;
    }
/**
 * Crea la escena de la pantalla del menú
 * @return devuelve la escena de la pantalla del menú
 */
    public Scene createMenuScene() {
```

```
        menuScene = new Scene();

        Sprite backgroundMenu = new Sprite(0, 0,
SplashActivity.CAMERA_WIDTH, SplashActivity.CAMERA_HEIGHT,
menu_background_region,
                engine.getVertexBufferObjectManager()){
            @Override
           protected void preDraw(GLState pGLState, Camera pCamera)
        {
               super.preDraw(pGLState, pCamera);
               pGLState.enableDither();
           }
        };

        menuScene.setBackground(new SpriteBackground(backgroundMenu));

        //button
        TiledSprite startButton1 = new
TiledSprite(SplashActivity.CAMERA_WIDTH/2-play1_region.getWidth()/2,
SplashActivity.CAMERA_HEIGHT/2-120, play1_region,
engine.getVertexBufferObjectManager()){
            @Override
    public boolean onAreaTouched(final TouchEvent pAreaTouchEvent, final
float pTouchAreaLocalX, final float pTouchAreaLocalY){
            switch(pAreaTouchEvent.getAction()){
            case TouchEvent.ACTION_DOWN:
                   this.setCurrentTileIndex(1);
                   break;
            case TouchEvent.ACTION_UP:
                   this.setCurrentTileIndex(0);
                   createGame1Scene();
                   setCurrentScene(AllScenes.GAME1);
                   break;
            }
            return true;
    }
    };
    menuScene.registerTouchArea(startButton1);
    menuScene.attachChild(startButton1);

    TiledSprite startButton2 = new
TiledSprite(SplashActivity.CAMERA_WIDTH/2-play1_region.getWidth()/2,
SplashActivity.CAMERA_HEIGHT/2, play2_region,
engine.getVertexBufferObjectManager()){
        @Override
    public boolean onAreaTouched(final TouchEvent pAreaTouchEvent, final
float pTouchAreaLocalX, final float pTouchAreaLocalY){
            switch(pAreaTouchEvent.getAction()){
            case TouchEvent.ACTION_DOWN:
                   this.setCurrentTileIndex(1);
                   break;
            case TouchEvent.ACTION_UP:
                   this.setCurrentTileIndex(0);
                   createGame2Scene();
                   setCurrentScene(AllScenes.GAME2);
                   break;
            }
```

```
            return true;
        }
        };
        menuScene.registerTouchArea(startButton2);
        menuScene.attachChild(startButton2);

        TiledSprite startButton3 = new
TiledSprite(SplashActivity.CAMERA_WIDTH/2-play1_region.getWidth()/2,
SplashActivity.CAMERA_HEIGHT/2+120, play3_region,
engine.getVertexBufferObjectManager()){
           @Override
       public boolean onAreaTouched(final TouchEvent pAreaTouchEvent, final
float pTouchAreaLocalX, final float pTouchAreaLocalY){
            switch(pAreaTouchEvent.getAction()){
            case TouchEvent.ACTION_DOWN:
                   this.setCurrentTileIndex(1);
                   break;
            case TouchEvent.ACTION_UP:
                   this.setCurrentTileIndex(0);
                   createGame3Scene();
                   setCurrentScene(AllScenes.GAME3);
                   break;
            }
            return true;
        }
        };
        menuScene.registerTouchArea(startButton3);
        menuScene.attachChild(startButton3);
        menuScene.setTouchAreaBindingOnActionDownEnabled(true);


        return menuScene;

        }
/**
 * Crea la escena del juego de pitch
 * @return devuelve la escna del juego de pitch
 */

       public Scene createGame1Scene() {
           //se crea la escena
           gameScene1 = new Scene();

           //font
           countingText = new Text(200f, 240f,
           fontDefault32Bold, "", 3,
           this.activity.getVertexBufferObjectManager());
           gameScene1.attachChild(countingText);

           //sprite
           final float centerX = (SplashActivity.CAMERA_WIDTH -
this.mHelicopterTextureRegion.getWidth()) / 2;
           final float centerY = (SplashActivity.CAMERA_HEIGHT -
this.mHelicopterTextureRegion.getHeight()) / 2;
           final AnimatedSprite helicopter = new AnimatedSprite(centerX,
centerY,
```

```
this.mHelicopterTextureRegion,this.activity.getVertexBufferObjectManager
() );

        helicopter.setScale(2);
        helicopter.animate(150);

        this.gameScene1.attachChild(helicopter);

        //comenzar grabacion
        audioRecord.startRecording();
        //
        final Sprite back = new Sprite(0,
SplashActivity.CAMERA_HEIGHT-this.mParallaxLayerBack.getHeight(),
mParallaxLayerBack,
                engine.getVertexBufferObjectManager()){

            @Override
            protected void preDraw(final GLState pGLState, final Camera
pCamera)
            {
                super.preDraw(pGLState, pCamera);
                pGLState.enableDither();
            }
                };

        Sprite mid = new Sprite(0, 80, mParallaxLayerMid,
                engine.getVertexBufferObjectManager());
        Sprite front = new Sprite(0,
SplashActivity.CAMERA_HEIGHT-this.mParallaxLayerFront.getHeight(),
mParallaxLayerFront,
                engine.getVertexBufferObjectManager());
        AutoParallaxBackground background = new AutoParallaxBackground(0, 0,
                0, 5);

        background.attachParallaxEntity(new ParallaxEntity(0,back));
        background.attachParallaxEntity(new ParallaxEntity(-3.0f,mid));
        background.attachParallaxEntity(new
ParallaxEntity(-10.0f,front));
        gameScene1.setBackground(background);
        gameScene1.setBackgroundEnabled(true);

        gameScene1.registerUpdateHandler(new IUpdateHandler() {

            @Override
            public void onUpdate(float pSecondsElapsed) {
                int f=frequency();
                if (f!=0){
                    if(f<60){
                        MoveYModifier moveYModifier = new
MoveYModifier(pSecondsElapsed,
                                helicopter.getY(), 480);
                        helicopter.registerEntityModifier(moveYModifier);
                    }
                    if(f>280){
                        MoveYModifier moveYModifier = new
MoveYModifier(pSecondsElapsed,
                                helicopter.getY(), 0);
```

```
                    helicopter.registerEntityModifier(moveYModifier);
                }
                if (60<f && f<280){

                MoveYModifier moveYModifier = new
MoveYModifier(pSecondsElapsed,
                        helicopter.getY(), Math.round(480-(f-60)*12/7));
                helicopter.registerEntityModifier(moveYModifier);
                }


            countingText.setText(String.valueOf(f));
            }

          }

        @Override public void reset() {}
        });

       return gameScene1;
    }
/**
 *  Crea la escena del juego de sonidos sonoros y sordos
 * @return devuelve la escena del juego de sonidos sonoros y sordos
 */
   public Scene createGame2Scene() {
       gameScene2 = new Scene();

       gameScene2.setBackground(new Background(0.5f, 0.6f, 0.2f));

       final float centerX = (SplashActivity.CAMERA_WIDTH -
this.mFaceTextureRegion.getWidth()) / 2;
       final float centerY = (SplashActivity.CAMERA_HEIGHT -
this.mFaceTextureRegion.getHeight()) / 2;
       final TiledSprite face= new TiledSprite(centerX, centerY,
this.mFaceTextureRegion,this.activity.getVertexBufferObjectManager() );

       face.setScale(2);



       this.gameScene2.attachChild(face);

       //comenzar grabacion
       audioRecord.startRecording();
       //

       gameScene2.registerUpdateHandler(new IUpdateHandler() {

           @Override
           public void onUpdate(float pSecondsElapsed) {
               int s=sonorosordo();
               if (s==2){
                   face.setCurrentTileIndex(0);
               }
               if (s==1){
                   face.setCurrentTileIndex(1);
```

```
                }
                if (s==0){
                    face.setCurrentTileIndex(2);
                }
            }

            @Override public void reset() {}
            });

        return gameScene2;
    }
/**
 * Crea la escena del juego de intensidad
 * @return devuelve la escena del juego de intensidad
 */
    public Scene createGame3Scene() {
        gameScene3 = new Scene();

        gameScene3.setBackground(new Background(0.7f, 0.1f, 0.2f));

        final float centerX = (SplashActivity.CAMERA_WIDTH -
this.mEnergiaTextureRegion.getWidth()) / 2;
        final float centerY = (SplashActivity.CAMERA_HEIGHT -
this.mEnergiaTextureRegion.getHeight()) / 2;
        final TiledSprite face= new TiledSprite(centerX, centerY,
this.mEnergiaTextureRegion,this.activity.getVertexBufferObjectManager()
);

        face.setScale(2);

        this.gameScene3.attachChild(face);

        //comenzar grabacion
        audioRecord.startRecording();
        //

        gameScene3.registerUpdateHandler(new IUpdateHandler() {

            @Override
            public void onUpdate(float pSecondsElapsed) {
                int s=energia();
                if (s==0){
                    face.setCurrentTileIndex(0);
                }
                if (s==1){
                    face.setCurrentTileIndex(1);
                }
            }

            @Override public void reset() {}
            });

        return gameScene3;
    }
/**
 *
 * @return devuelve la escena actual
```

```
 */
    public AllScenes getCurrentScene() {
        return currentScene;
    }

/**
 * Establece una de las escenas de la aplicación
 * @param currentScene es la escena actual
 */
    public void setCurrentScene(AllScenes currentScene) {
        this.currentScene = currentScene;
        switch (currentScene) {
        case SPLASH:
            break;
        case MENU:
            engine.setScene(menuScene);
            break;
        case GAME1:
            engine.setScene(gameScene1);
            break;
        case GAME2:
            engine.setScene(gameScene2);
            break;
        case GAME3:
            engine.setScene(gameScene3);
            break;

        default:
            break;
        }
    }
/**
 *  Calcula la intensidad de las muestras de audio recogidas
 * @return devuelve 0 si es inferior al umbral de intesnidad y 1 en caso
contrario
 */
private int energia(){


    while( true )
    {

        audioRecord.read(buffer, 0, LongVentana);

        int s=0;
        double UMBRAL_VOZ_ENER=65;
        double [] Elog = new double[LongVentana/2];
        double [] elin = new double[LongVentana/2];
        double E=0;
        double eps=1e-12;

        //Se calcula la energía logarítmica del segmento
        for (int i = 0; i < LongVentana/2; i++){
        elin[i]=buffer[i]*buffer[i];
        Elog[i]=10*Math.log10( eps+elin[i] );
        E = E + Elog[i];
```

```
        }
        E = E/(LongVentana/2);

        if (E>UMBRAL_VOZ_ENER){
         s=1;

        }
        return s;
    }
}

/**
 *  Calcula si los datos de audio son sonoros o sordos o si la intensidad
es inferior al umbral definido
 * @return devulve 0 si no supera el umbral de intesnidad, 1 si es sordo y
2 si es sonoro
 */
private int sonorosordo(){


        while( true )
        {
            audioRecord.read(buffer, 0, LongVentana);

            int s=0;
            double UMBRAL_VOZ_ENER=63.5;
            double UMBRAL_VOZ_TCC=0.25;
            double [] Elog = new double[LongVentana/2];
            double [] elin = new double[LongVentana/2];
            double E=0;
            double eps=1e-12;
            double TCC;
            double numZC=0;

            //Se calcula la energía logarítmica del segmento
            for (int i = 0; i < LongVentana/2; i++){
             elin[i]=buffer[i]*buffer[i];
             Elog[i]=10*Math.log10( eps+elin[i] );
             E = E + Elog[i];

            }
            E = E/(LongVentana/2);
            //Se calcula la tasa de cruces por cero

            for (int i=0; i<(LongVentana/2)-1; i++){
                   if((buffer[i]>=0 && buffer[i+1]<0) || (buffer[i]<0 &&
buffer[i+1]>=0)){
                          numZC++;
                   }
            }
            TCC=numZC/1024;

            if (E>UMBRAL_VOZ_ENER & TCC<UMBRAL_VOZ_TCC){
             s=1;}

            if (E>UMBRAL_VOZ_ENER & TCC>UMBRAL_VOZ_TCC){
             s=2;}
```

```
            return s;
        }

}
/**
 * Calcula la frecuencia de los datos de audio
 * @return devuelve la frecuencia
 */

private int frequency(){


        while( true )
        {
            audioRecord.read(buffer, 0, LongVentana);

            int Tmin, Tmax;
            int UMBRAL_ENER=20;
            int tam_fft;
            int i_max=0;
            double max=0;
            int i_max2=0;
            double max2=0;
            double mean=0;
            double total=0;
            int T;
            int freq;
            int m = (LongVentana/2) / 2;
            double r;
            double pi = Math.PI;
            double[] w = new double[LongVentana/2];
            double[] ventana = new double[LongVentana/2];
            double[] cep_alta = new double[LongVentana/4];
            double[] cep_energia = new double[LongVentana/4];

            Tmin=(int)Math.round(0.002*SAMPPERSEC);
            Tmax=(int)Math.round(0.015*SAMPPERSEC);
            //fft rapidas
            tam_fft=(int)Math.pow( 2, Math.ceil(  Math.log(LongVentana/2) /
Math.log(2)));
            //ventana hamming
            r = pi / m;
            for (int n = -m; n < m; n++){
                w[m + n] = 0.54f + 0.46f * Math.cos(n * r);
            }
          //ventana es la voz enventanda usando hamming
            for (int i = 0; i < LongVentana/2; i++){
             ventana[i]=w[i]*buffer[i];
            }
            //cepstrum
            DoubleFFT_1D fftDo = new DoubleFFT_1D(tam_fft);
            double[] fft = new double[tam_fft*2];
            System.arraycopy(ventana, 0, fft, 0, tam_fft);
            fftDo.realForward(fft);
            for (int i = 0; i < tam_fft*2; i++){
             fft[i]=Math.log(Math.abs(fft[i]));
```

```
            }
            fftDo.realInverse(fft,true);
            //cep-alta
            for (int i = Tmin; i < tam_fft/4; i++){
             cep_alta[i]=fft[i];
            }
            //cep_energia
            for (int i = Tmin; i < tam_fft/4; i++){
             cep_energia[i]=cep_alta[i]*cep_alta[i];
            }
            //max y i_max
            for (int i = Tmin; i < Tmax-Tmin; i++){
             if (max<cep_energia[i]){
                 max=cep_energia[i];
                 i_max=i;
             }
            }
            //media cep_energia
            for (int i = Tmin; i < tam_fft/4; i++){
             total+=cep_energia[i];
             mean=total/((tam_fft/4)-Tmin);
            }
            //Se buscan máximos anteriores (de índice menor) que tengan
amplitud suficiente,
            // quedándonos con el último que encontremos (el primer máximo que
supera el umbral).
            if (max>UMBRAL_ENER*mean){
             for (int i = Tmin; i < i_max-Tmin; i++){
                 if (max2<cep_energia[i]){
                     max2=cep_energia[i];
                     i_max2=i;
                 }
                }

             while (max2>UMBRAL_ENER*mean){
                  i_max=i_max2;
                  for (int i = Tmin; i < i_max2-Tmin; i++){
                        if (max2<cep_energia[i]){
                            max2=cep_energia[i];
                            i_max2=i;
                        }
                     }
             }
             // i_max+Tmin-1 es el numero de muestra dentro del vector cepstrum
             T=i_max+Tmin-1;
                if (T!=0){
                   freq=Math.round(SAMPPERSEC/T);
                   }

                else{
                   freq=0;
                   }
            }

            else{
             freq=0;
            }
```

```
            return freq;
        }
    }
}
```

Referencias

[1] AULA SALUD, *Desarrollo apps de salud para IOS y Android*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://www.aula-salud.com/consultoria/desarrollo-apps

[2] BELLOCH, Consuelo. [9 de Abril de 2014]. *Recursos tecnológicos para la intervención en trastornos del habla y voz*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://www.uv.es/bellochc/logopedia/NRTLogo5.wiki

[3] TORRES Begoña, *Anatomía funcional de la voz*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://www.medicinadelcant.com/cast/1.pdf

[4] ZONIC STUDIO, *Producción de la voz*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: https://sites.google.com/site/zonicstudio/canto/produccion-de-la-voz

[5] COBETA Ignacio, NÚÑEZ Faustino, FERNÁNDEZ Secundino. En: *Patología de la voz*. Barcelona. Edición: ICG Marge, 2013.

[6] INSTITUO NACIONAL DEL CÁNCER, *Cáncer de laringe,* [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://www.cancer.gov/espanol/pdq/tratamiento/laringe/Patient/page1

[7] IBÁÑEZ Arthur, *Resonadores humanos,* [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://www.arthuribanez.com/resonadores/

[8] MORRISON Murray, RAMMAGE Linda, *Tratamiento de los trastornos de la voz*. Barcelona. Edición: Masson S.A., 1996.

[9] MINTLEAF SOFTWARE INC., *Sonneta Voice Monitor*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://mintleafsoftware.com/voice-monitor.html

[10] CTS INFORMÁTICA, *Voxtraining-Aviao*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://www.ctsinformatica.com.br/#voxTrainingAviao.html

[11] CTS INFORMÁTICA, *Voxtraining-Gaivota*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://www.ctsinformatica.com.br/#voxTrainingGaivota.html

[12] MICRO VIDEO CORPORATION, *Sample Fun & Games Screen*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://www.videovoice.com/vv_fgrlgl.htm

[13] DR. SPEECH, *Speech Therapy Standard Version*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://www.drspeech.com/SpeechTherapy5.html

[14] KAYPENTAX, *Voice Games, Model 5167B*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://www.kayelemetrics.com/index.php?option=com_product&Itemid=3&controller=product&task=learn_more&cid%5B%5D=53

[15] SPEECH THERAPISTS DON'T GET APPLES, *Slp Android apps,* [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://rwslp.wordpress.com/slp-android-apps/

[16] GOOGLE PLAY, *SpaceWilli*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: https://play.google.com/store/apps/details?id=de.hoch3.spacewilli.paid&hl=es

[17] LITTLE BEE SPEECH, *Articulation Station Pro*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://littlebeespeech.com/articulation_station_pro.php

[18] EDUCATIONALAPPSTORE LTD., *Speech Companion Speech Therapy*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://www.educationalappstore.com/app/speech-companion-speech-therapy

[19] CORONA LABS INC., *Corona SDK*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://coronalabs.com/products/corona-sdk/

[20] COCOS2D-X, *Cosos2d-x*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://www.cocos2d-x.org/

[21] MARIO ZECHNER, *Libgdx Goals and Features*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://libgdx.badlogicgames.com/

[22] MARMALADE TECHNOLOGIES LTD., *Marmalade C++ SDK*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: https://www.madewithmarmalade.com/

[23] ANDENGINE, *Andengine*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://www.andengine.org/

[24] ANDROID INC., *Android Development Tools*, [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: http://developer.android.com/tools/help/adt.html

[25] DARCEY Lauren y CONDER Shane. "Configurar el entorno de desarrollo Android". En: *Programación Android 4.* Madrid: Ediciones Anaya Multimedia, 2012. Pág. 75

[26] SCHROEDER Jayme y BROYLES Brian. En: *Andengine for Android Game Development Cookbook*. Birmingham: Packt Publishing, 2013.

[27] WENDYKIER, Piotr. *JTransforms* [en línea], [Consulta: 14 de Agosto de 2014]. Disponible en la web: https://sites.google.com/site/piotrwendykier/software/jtransforms

[28] GRAMLICH Nicolas. *AndEngineExamples*, [en línea], [Consulta: 18 de Agosto de 2014]. Disponible en la web: https://github.com/nicolasgramlich/AndEngineExamples

[29] FARLEX, Inc. *The Free Dictionary*, [en línea], [Consulta: 18 de Agosto de 2014]. Disponible en la web: http://www.thefreedictionary.com/

[30] DICCIONARIO MÉDICO, *Diccionario Médico*, [en línea], [Consulta: 18 de Agosto de 2014]. Disponible en la web: http://www.diccionariomedico.net/

[31] WORDREFERENCE, *WordReference*, [en línea], [Consulta: 18 de Agosto de 2014]. Disponible en la web: http://www.wordreference.com/

[32] TALLIS Jaime, SOPRANO Ana María, En: *Neuropediatría, Neuropsicología y Aprendizaje*, Buenos Aires: Nueva Visión, 1991.

[33] WIKIPEDIA, *Dithering*, [en línea], [Consulta: 18 de Agosto de 2014]. Disponible en la web: http://es.wikipedia.org/wiki/Dithering